MIRtoolbox 1.4

User's Manual

Olivier Lartillot
Finnish Centre of Excellence in Interdisciplinary Music Research
Swiss Center for Affective Sciences
May, 30th, 2012

TABLE OF CONTENTS

1. Introduction 6

Conditions of Use 6

Please Register 6

Documentation and Support 6

Background 7

MIRtoolbox Objectives 8

MIRtoolbox Features 9

Installation 12

Help and demos 14

MIRtoolbox Interface 15

2. Basic Operators 21

miraudio 21

mirframe 24

mirfilterbank 27

mirenvelope 31

mirspectrum 38

mircepstrum 46

mirautocor 50

* 57

mirflux 58

mirsum 61

mirpeaks 63

mirsegment 68

mirplay 71

mirsave 73

mirlength 75

3. Feature Extractors 76

3.1. Dynamics 76

mirrms 76

mirsegment(..., 'RMS') 78

mirlowenergy 79

3.2. Rhythm 81

mirfluctuation 81

mirbeatspectrum 83

mironsets 84

mireventdensity 91

mirtempo 92

mirpulseclarity 96

3.3. Timbre 99

mirattacktime 99

mirattackslope 101

mirattackleap 103

mirzerocross 104

mirrolloff 106

mirbrightness 107

mirmfcc 108

mirroughness 110

mirregularity 112

3.4. Pitch 113

mirpitch 113

mirmidi 118

mirinharmonicity 119

3.5. Tonality 121

mirchromagram 121

mirkeystrength 125

mirkey 127

mirmode 129

mirkeysom 131

mirtonalcentroid 133

mirhcdf 134

mirsegment(..., 'HCDF') 135

4. High-level features 136

4.1. Structure and form 136

mirsimatrix 136

mirnovelty 142

mirsegment(..., 'Novelty') 145

4.2. Statistics 147

mirmean 147

mirstd 147

mirstat 148

mirhisto 149

mirzerocross 150

mircentroid 151

mirspread 152

mirskewness 153

mirkurtosis 155

mirflatness 156

mirentropy 157

mirfeatures 158

mirmap 161

4.3. Predictions 164

miremotion 164

mirclassify 171

mircluster 172

4.4. Similarity and Retrieval 174

mirdist 174

mirquery 175

4.5. Exportation 176

mirgetdata 176

mirexport 178

5. Advanced use of MIRtoolbox 179

5.1. Interface preferences 179

5.2. get 180

5.3. Memory management 181

References 186

1. INTRODUCTION

Conditions of Use

The Toolbox is free software; you can redistribute it and/or modify it under the terms of version 2 of GNU General Public License as published by the Free Software Foundation.

When *MIRtoolbox* is used for academic research, we would highly appreciate if scientific publications of works partly based on *MIRtoolbox* cite one of the following publications:

Olivier Lartillot, Petri Toiviainen, "A Matlab Toolbox for Musical Feature Extraction From Audio", *International Conference on Digital Audio Effects*, Bordeaux, 2007.

Olivier Lartillot, Petri Toiviainen, Tuomas Eerola, "A Matlab Toolbox for Music Information Retrieval", in C. Preisach, H. Burkhardt, L. Schmidt-Thieme, R. Decker (Eds.), *Data Analysis, Machine Learning and Applications*, Studies in Classification, Data Analysis, and Knowledge Organization, Springer-Verlag, 2008.

For commercial use of *MIRtoolbox*, please contact the authors.

Please Register

Please register to the MIRtoolbox announcement list.

This will allow us to estimate the number of users, and this will allow you in return to get informed on the new major releases (including critical bug fixes).

Documentation and Support

The URL of *MIRtoolbox* website is www.jyu.fi/music/coe/materials/mirtoolbox

MIRtoolbox Discussion List

A discussion list is also available:

- To subscribe, send an empty mail with 'Subscribe' as subject to mirtoolbox-request@freelists.org
- The archive is available here.

MIRtoolbox Tweets

Get informed of the day-to-day advance of the project (bug reports, bug fixes, new features, new topics, etc.) by following @mirtoolbox.

Tutorial Video

Video recordings of a tutorial given during SMC09 are available on *YouTube*.

Background

About the Authors

Olivier Lartillot, Petri Toiviainen and Tuomas Eerola are members of the *Finnish Centre of Excellence in Interdisciplinary Music Research*, University of Jyväskylä, Finland. Olivier Lartillot is also member of the *Swiss Center for Affective Sciences*, University of Geneva, Switzerland.

The development of the toolbox has benefitted from productive collaborations with:

- partners of the *Brain Tuning* project (Marco Fabiani, Jose Fornari, Anders Friberg, Roberto Bresin, ...),
- colleagues from the *Finnish Centre of Excellence in Interdisciplinary Music Research* (Pasi Saari, Martin Hartmann, Vinoo Alluri, Rafael Ferrer, Marc Thompson, ...),
- colleagues from the *Swiss Center for Affective Sciences* (Didier Grandjean, Klaus Scherer, Kim Eliard, Carolina Labbé, Johanna Wiebke Trost, Donato Gereghetti, ...),
- students of the MMT master program,
- external collaborators: Jakob Abeßer (Fraunhofer IDMT), Donald Glowinski (infoMus, Casa Paganini), Thomas Wosch and associates (MEM, FHWS), Cyril Laurier and Emilia Gomez, (MTG-UPF),
- active users of the toolbox, participating in particular to the discussion list,
- participants of the SMC Summer School 2007, ISSSM 2007, ISSSCCM 2009, USMIR 2010, ISSAS 2011.

Tuning the Brain for Music

MIRtoolbox has been developed within the context of a Europeen Project called "*Tuning the Brain for Music*", funded by the NEST (New and Emerging Science and Technology) program of the European Commission. The project, coordinated by Mari Tervaniemi from the Cognitive Brain Research Unit of the Department of Helsinki, is dedicated to the study of music and emotion, with collaboration between neurosciences, cognitive psychology and computer sci-

ence. One particular question, studied in collaboration between the Music Cognition Team of the University of Jyväskylä and the Music Acoustics Group of the KTH in Stockholm, is related to the investigation of the relation between musical features and music-induced emotion. In particular, we would like to know which musical parameters can be related to the induction of particular emotion when playing or listening to music. For that purpose, we needed to extract a large set of musical features from large audio data-bases, in order to perform in a second step a statistical mapping between the diverse musical parameters and musical materials with listeners' emotional ratings. This requires in particular a management of the interdependencies between the diverse features – in order to avoid having to recompute the same operations again and again – and also a control of the memory costs while analyzing the databases.

Music, Mind, Technology

The Music Cognition Team has recently introduced a new master degree, called *Music Mind Technology* (MMT). The *Music Information Retrieval* course, taught by Petri Toiviainen, Vinoo Alluri and myself, offers an overview of computer-based research for music analysis and in particular musical feature extraction. For the hands-on sessions, we wanted the student to be able to try by themselves the different computational approaches using *Matlab*. As many of them did not have much background in this programming environment, we decided to design a computational environment for musical feature extraction aimed at both expert and non-expert of *Matlab*.

MIRtoolbox Objectives

Due to the context of development of this toolbox, we elaborated the following specifications:

General Framework

MIRtoolbox proposes a large set of musical feature extractors.

Modular Framework

MIRtoolbox is based on a set of building blocks that can be parametrized, reused, reordered, etc.

Simple and Adaptive Syntax

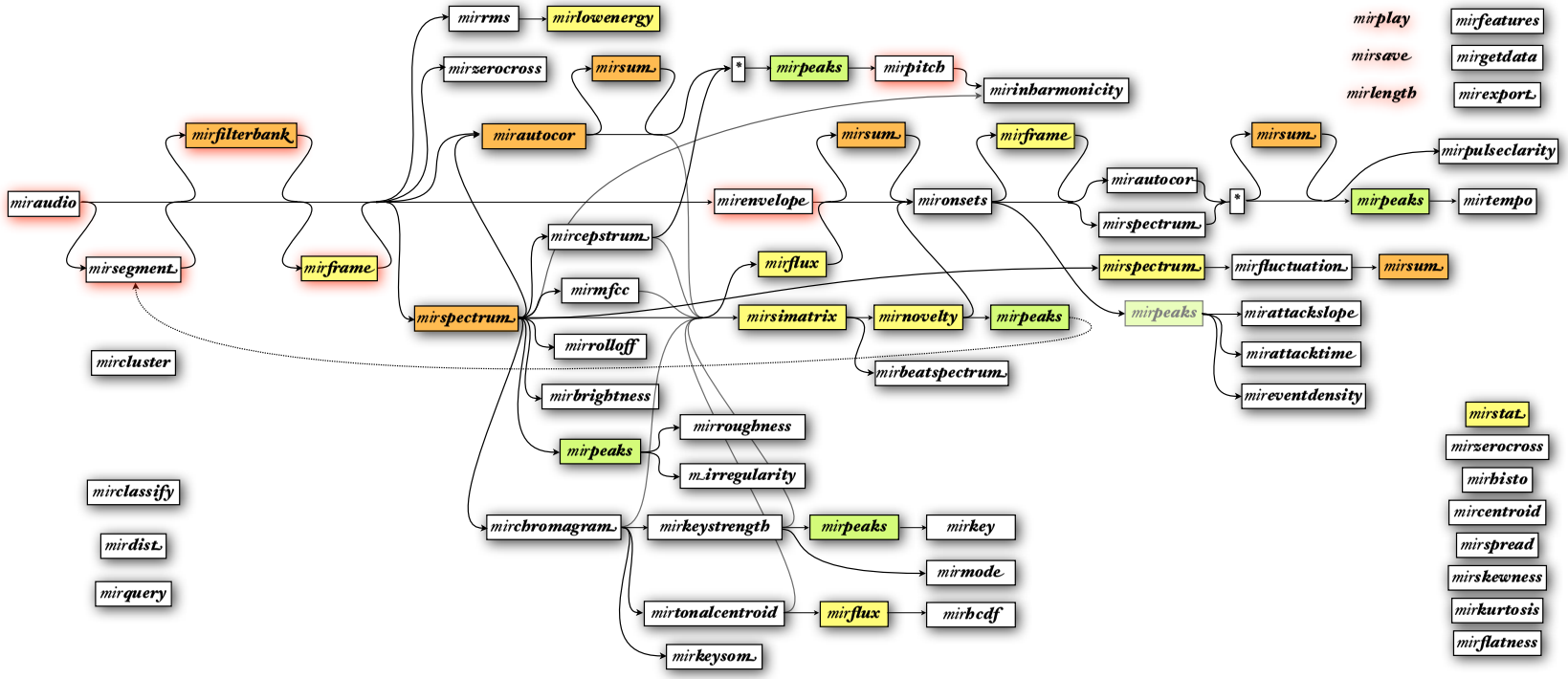
Users can focus on the general design, *MIRtoolbox* takes care of the underlying laborious tasks.

Free Software, Open Source

The idea is to propose to capitalize the expertise of the research community, and to offer it back to the community and the rest of us.

MIRtoolbox Features

MIRtoolbox includes around 50 audio and music features extractors and statistical descriptors. A brief overview of most of the features can be seen in the following figure.



Synthetic overview of the features available in MIRtoolbox 1.2

MIRtoolbox Reliances

Required commercial products

MIRtoolbox requires the ***Matlab*** environment, **version 7,** and does not work very well with previous versions of *Matlab*. This is due in particular to the fact *MIRtoolbox* relies on multi-dimensional arrays and multiple outputs, which seem to be features introduced by version 7.

MIRtoolbox also requires that the ***Signal Processing Toolbox***, one of the optional sub-packages of *Matlab*, be properly installed. But actually, a certain number of operators can adapt to the absence of this toolbox, and can produce more or less reliable results. But for serious use of *MIRtoolbox*, we strongly recommand a proper installation of the *Signal Processing Toolbox*.

Free softwares included in the *MIRtoolbox* distribution

MIRtoolbox includes in its distribution several other freely available toolboxes, that are used for specific computations.

- The *Auditory Toolbox*, by Malcolm Slaney (1998), is used for Mel-band spectrum and MFCC computations, and Gammatone filterbank decomposition.
- The *Netlab* toolbox, by Ian Nabney (2002), where the routines for Gaussian Mixture Modeling (GMM) is used for classification (*mirclassify*).
- Finally, the SOM toolbox, by Esa Alhoniemi and colleagues (Vesanto, 1999), where only a routine for clustering based on *k*-means method is used, in the *mircluster* function.

Code integrated as part of GPL project

MIRtoolbox license is based on GPL 2.0. As such, it can integrate codes from other GPL 2.0 projects, as long as their origins are explicitly stated.

- codes from the *Music Analysis Toolbox* by Elias Pampalk (2004), related to the computation of Terhardt outer ear modeling, Bark band decomposition and masking effects. (GPL 2.0)
- implementation of Earth Mover Distance written by Yossi Rubner and wrapped for Matlab by Simon Dixon.
- *openbdf* and *readbdf* script by T.S. Lorig to read BDF files, based on *openedf* and *readedf* by Alois Schloegl.

Code integrated with BSD license

- *mp3read* for *Matlab* by Dan Ellis, which calls the *mpg123* decoder and the *mp3info* scanner.

- *aiffread* for Matlab by Kenneth Eaton

Installation

To install *MIRtoolbox* in your *Matlab* environment, move the main *MIRtoolbox* folder to the location of your choice in your computer (for instance, in your *Matlab* "*toolbox*" folder, if you have administrative rights to modify it). Then open the "*Set Path*" environment available in *Matlab File* menu, click on "*Add with Subfolders...*", browse into the file hierarchy and select the main *MIRtoolbox* folder, then click "*Open*". You can then "*Save*" and "*Close*" the *Set Path* environment.

Update

If you replace an older version of *MIRtoolbox* with a new one, please update your *Matlab* path using the following command:

rehash toolboxcache

Update also the class structure of the toolbox, either by restarting Matlab, or by typing the following command:

clear classes

MP3 reader for Mac OS X 64-bits platform

If you are running *Matlab* on a *Mac OS X* 10.6 or beyond and with *Matlab* release 2009 or beyond, the binaries used for reading MP3 files (*mpg123* and *mp3info*) needs to be in 64-bits format (with the *mexmaci64* file extension). Unfortunately, it seems that the *mpg123.mexmaci64* and *mp3info.mexmaci64* executable we provided in the *MIRtoolbox* distribution cannot be used directly on other computers, so you need to install those binaries by yourselves on each separate computer by doing the following:

- Install Apple's *Xcode*:
  - If you use Max OS X 10.7, you can download it from free on the Mac App Store.
  - If you use Mac OS X 10.6, you need to be (freely) registered as an Apple Developer. We suggest to download *Xcode* 3.2.6, as it is the latest free version available.
- Install *MacPorts*.
- Check that your *MacPorts* is up-to-date by executing in the Terminal:

sudo port -v selfupdate

(You need to authenticate as an administrative user.)

- Install *mpg123* and *mp3info* via *MacPorts* by executing in the Terminal:

sudo port install mpg123

sudo port install mp3info

(Each of these two installations might take some time.)

- Once both installations are completed, you should obtain among others two Unix executable files called *mpg123* and *mp3info*, probably located at the address */opt/local/bin*.
- Create a copy of these files that you rename *mpg123.mexmaci64* and *mp3info.mexmaci64*, and place these two renamed files in a folder whose path is included in Matlab. You can for instance place them in your *MIRtoolbox* folder, which already contains Unix executable *mpg123.mexmaci* and *mp3info.mexmaci*, which correspond to the 32-bit platform. If there already exists files called *mpg123.mexmaci64* and *mp3info.mexmaci64*, you can replace those previous files with the new ones you compiled yourself.

Help and demos

To get an overview of the functions available in the toolbox, type:

help mirtoolbox

A short documentation for each function is available using the same *help* command. For instance, type:

help miraudio

Demos

Examples of use of the toolbox are shown in the *MIRToolboxDemos* folder:

- *mirdemo*
- *demo1basics*
- *demo2timbre*
- *demo3tempo*
- *demo4segmentation*
- *demo5export*
- *demo6curves*
- *demo7tonality*
- *demo8classification*
- *demo9retrieval*

MIRtoolbox Interface

Basic syntax

All functions are preceded by the *mir-* prefix in order to avoid conflicts with other Matlab functions. Each function is related to a particular data type: for instance, *miraudio* is related to the loading, transformation and display of audio waveform. An audio file, let's say a WAV file of name *mysong.wav*, can be loaded simply by writing the command:

*miraudio('**mysong.wav**')*

The extension of the file can be omitted:

*miraudio(**'mysong'**)*

Operations and options to be applied are indicated by particular keywords, expressed as arguments of the functions. For instance, the waveform can be centered using the *'Center'* keyword:

*miraudio('mysong', **'Center'**)*

which is equivalent to any of these parameters:

*miraudio('mysong', 'Center', **'yes'**)*

*miraudio('mysong', 'Center', **'on'**)*

*miraudio('mysong', 'Center', **1**)*

whereas the opposite set of parameters

*miraudio('mysong', 'Center', **'no'**)*

*miraudio('mysong', 'Center', **'off'**)*

*miraudio('mysong', 'Center', **0**)*

are not necessary in the case of the *'Center'* options as it is toggle off by default in *miraudio*.

It should be noted also that keywords are not case-sensitive:

miraudio('mysong', 'center', 'YES')

Other options accept numerical particular parameters. For instance, an audio waveform can be resampled to any sampling rate, which is indicated by a value in Hertz (Hz.) indicated after the *'Sampling'* keyword. For instance, to resample at 11025 Hz., we just write:

*miraudio('mysong', **'Sampling', 11025**)*

Finally the different options can be combined in one single command line:

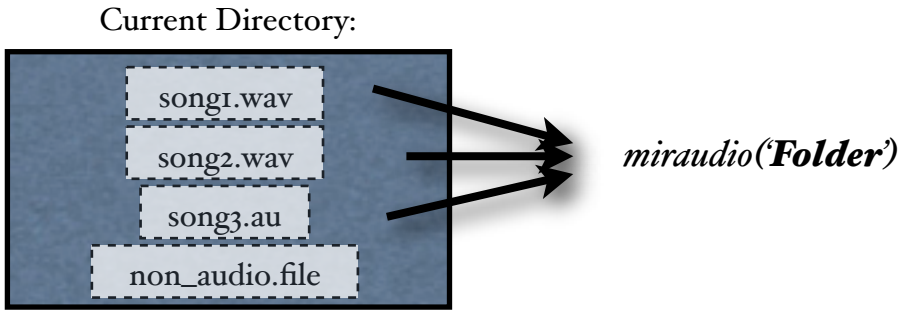
*miraudio('mysong', **'Center', 'Sampling', 11025**)*

Batch analysis

- Folder of files can be analyzed in exactly the same way. For that, the file name, which was initially the first argument of the functions, can be replaced by the ***'Folder'*** keyword. For instance, a folder of audio files can be loaded like this:

*miraudio('**Folder**')*

Only audio files in the WAV and AU formats are taken into consideration, the other files are simply ignored:



Automatic analysis of a batch of audio files using the 'Folder' keyword

- Subfolders can be analyzed recursively as well, using the '***Folders***' keyword:

*miraudio('**Folders**')*

- Alternatively, the list of audio files (with their respective path) can be stored in successive lines of a TXT file, whose name (and path) can be given as input to *miraudio*:

*miraudio('myfilenames**.txt**')*

- As another alternative, the list of audio files (with address relative to the current directory) can be given in a cell array as first input to *miraudio*, for instance:

miraudio({'song1', 'song2', 'song3'})

Output format

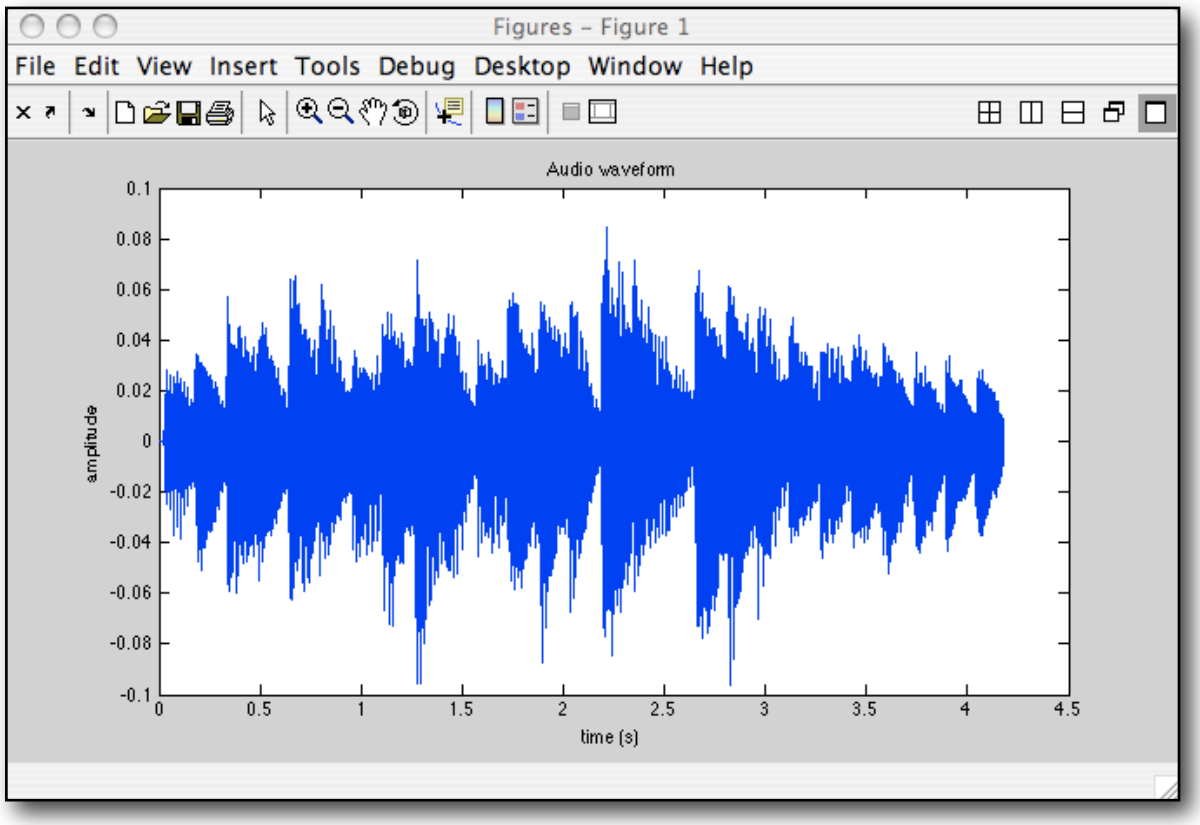
After entering one command, such as

miraudio('mysong')

the computation is carried out, and when it is completed, a text is written in the Command Window:

```
ans is the Audio waveform related to file mysong.wav, of sampling
rate 44100 Hz.

Its content is displayed in Figure 1.
```

And a graphical representation of the result is displayed in a figure:



Display of a miraudio *object.*

The display of the figures and the messages can be avoided, if necessary, by adding a semi-colon at the end of the command:

miraudio('mysong');

The actual output is stored in an object, hidden by default to the users, which contains all the information related to the data, such as the numerical values of the waveform amplitudes, the temporal dates of the bins, the sampling rate, the name of the file, etc. In this way we avoid the traditional interface in Matlab, not quite user-friendly in this respect, were results are directly displayed in the Command Window by a huge list of numbers.

It is not possible to display MIRtoolbox results in the Matlab Variable Editor. If you try visualizing a MIRtoolbox variable listed in your Workspace window, for instance the audio waveform in the previous example, you get the following text in the Variable Editor:

```
val is the Audio waveform related to file mysong.wav, of sampling
rate 44100 Hz.

To display its content in a figure, evaluate this variable directly
in the Command Window.
```

MULTIPLE FILE OUTPUT

If we now analyze a folder of file:

*miraudio('**Folder**')*

the results related to each audio file is displayed in a different figure, and messages such as the following ones are displayed in the Command Window:

```
ans(1) is the Audio waveform related to file song1.wav, of sampling
rate 44100 Hz.

Its content is displayed in Figure 1.

ans(2) is the Audio waveform related to file song2.wav, of sampling
rate 22050 Hz.

Its content is displayed in Figure 2.

ans(3) is the Audio waveform related to file song3.au, of sampling
rate 11025 Hz.

Its content is displayed in Figure 3.
```

and so on.

And the actual output is stored in one single object, that contains the information related to all the different audio files.

Threading of data flow

The result of one operation can be used for subsequent operations. For that purpose, it is better to store each result in a variable. For instance, the audio waveform(s) can be stored in one variable a:

a = *miraudio('mysong');*

Then the spectrum, for instance, related to the audio waveform can be computed by calling the function *mirspectrum~* using simply the a variable as argument:

s = *mirspectrum(**a**)*

In this way, all the information necessary for the computation of the spectrum can be retrieved from the hidden object, associated to the variable a, that contains the complex encapsulated data.

Alternatively, the spectrum can be directly computed from a given audio file by indicating the file name as argument of the *mirspectrum~* function:

s = *mirspectrum(**'mysong'**)*

This second syntax, more compact, is generally recommended, because it avoids the decomposition of the computation in several steps (*a*, then *s*, etc.), which might cause significant problems for long audio files or for folder of files. We will see in section 5.3 how to devise more subtle datacharts that take into account memory management problems in a more efficient way.

Successive operations on one same data format

When some data has been computed on a given format, let's say an audio waveform using the miraudio function:

a = *miraudio('mysong');*

it is possible to apply options related to that format in successive step. For instance, we can center the audio waveform in a second step:

a = *miraudio(**a**, 'Center');*

which could more efficiently be written in one single line:

a = *miraudio(**'mysong', 'Center'**);*

Numerical data recuperation

The numerical data encapsulated in the output objects can be recuperated if necessary. In particular, the main numerical data (such as the amplitudes of the audio waveform) are obtained using the *mirgetdata* command:

mirgetdata*(a)*

the other related informations are obtained using the generic ***get*** method. For instance, the sampling rate of the waveform a is obtained using the command:

get*(a, 'Sampling')*

More detailed description of these functions will be described in section 5, dedicated to advance uses of *MIRtoolbox*.

2. BASIC OPERATORS

MIRtoolbox basic operators concern the management of audio waveforms (*miraudio*, *mirsave*), frame-based analysis (*mirframe*, *mirflux*), periodicity estimation (*mirautocor*, *mirspectrum*, *mircepstrum*), operations related more or less to auditory modeling (*mirenvelope*, *mirfilterbank*), peak picking (*mirpeaks*) and sonification of the results (*mirplay*).

miraudio

Audio waveform

As explained previously, this operator basically loads audio files, displays and performs operations on the waveform.

Accepted input formats

- **file name**: The accepted file formats are WAV, MP3, AIFF and AU formats, as the loading operations are based on the *Matlab wavread* and *auread* functions, on Dan Ellis' *mp3read* and on Kenneth Eaton's *aiffread*.
- ***miraudio* object**: for further transformations.
- ***Matlab* array**: It is possible to import an audio waveform encoded into a *Matlab* column vector, by using the following syntax:

miraudio(v, sr)

where *v* is a column vector and *sr* is the sampling rate of the signal, in Hz. The default value for *sr* is 44100 Hz.

Transformation options

- *miraudio(..., '**Mono**', 0)* does not perform the default summing of channels into one single mono track, but instead stores each channel of the initial sound file separately.
- *miraudio(..., '**Center**')* centers the waveform.
- *miraudio(..., '**Sampling**', r)* resamples at sampling rate *r* (in Hz). It uses the *resample* function from *Signal Processing Toolbox*.
- *miraudio(..., '**Normal**')* normalizes with respect to RMS energy (cf. *mirrms*).

- *miraudio(..., 'Frame', w, wu, h, hu)* decomposes into frames. Cf. *mirframe* for an explanation of the arguments (units can be omitted here as well). Default parameters: same as in *mirframe*, i.e., 50 ms and half-overlapping.

Extraction options

- *miraudio(..., 'Extract', t1, t2, u,f)* extracts the signal between the dates *t1* and *t2*, expressed in the unit *u*.
  - Possible units *u* = *'s'* (seconds, by default) or *u* = *'sp'* (sample index, starting from 1).
  - The additional optional argument *f* indicates the referential origin of the temporal positions. Possible values for f:
    - '*Start*' (by default),
    - '*Middle*' (of the sequence),
    - '*End*' of the sequence.

When using '*Middle*' or '*End*', negative values for *t1* or *t2* indicate values before the middle or the end of the audio sequence. For instance: *miraudio(..., 'Extract'*, -1, +1, *'Middle')* extracts one second before and after the middle of the audio file.

  - Alternative keyword: *'Excerpt'*.
- *miraudio(..., 'Trim')* trims the pseudo-silence beginning and end off the audio file.
  - *miraudio(..., 'TrimThreshold', t)* specifies the trimming threshold *t*. Silent frames are frames with RMS energy below *t* times the medium RMS of the whole audio file. Default value: *t* = 0.06.
  - Instead of *'Trim'*, *'TrimStart'* only trims the beginning of the audio file, whereas *'TrimEnd'* only trims the end.
- *miraudio(..., 'Channel', c)* or *miraudio(.., 'Channels', c)* selects the channels indicated by the (array of) integer(s) *c*.

Labeling option

miraudio(..., 'Label', lb) labels the audio signals following the name of their respective audio files. *lb* is one number, or an array of numbers, and the audio signals are labelled using the substring of their respective file name of index lb. If *lb* =0, the audio signal(s) are labelled using the whole file name.

miraudio('Folder', 'Label', lb)	*song1g.wav*	*song2g.wav*	*song3b.au*

lb = 6	*'g'*	*'g'*	*'b'*
lb = [5 6]	*'1g'*	*'2g'*	*'3b'*
lb = {'good', 'bad'}	*'good'*	*'bad'*	*'good'*

Example of labelling of a folder of audio files

The labeling is used for classification purposes (cf. *mirclassify* and *mirexport*).

Summation

Audio signals can be superposed using the basic *Matlab* summation operators (+). For instance let's say we have two sequences:

```
a1= miraudio('melody.wav');
```

```
a2= miraudio('accompaniment.wav');
```

Then the two sequences can be superposed using the command:

```
a = a1+a2
```

When superposing *miraudio* objects, the longest audio are no more truncated, but on the contrary the shortest one are prolonged by silence. When audio have different sampling rates, all are converted to the highest one.

Accessible Output

cf. §5.2 for an explanation of the use of the *get* method. Specific fields:

- '***Time***': the temporal positions of samples (same as '*Pos*'),
- '***Centered***': whether the waveform has been centered (1) or not (0),
- '***NBits***': the number of bits used to code each sample,
- '***Label***': the label associated to each audio file.

mirframe

Frame Decomposition

The analysis of a whole temporal signal (such as an audio waveform in particular) leads to a global description of the average value of the feature under study. In order to take into account the dynamic evolution of the feature, the analysis has to be carried out on a short-term window that moves chronologically along the temporal signal. Each position of the window is called a frame.

Flowchart Interconnections

mirframe accepts as input any temporal object:

- an audio waveform ***miraudio***,
- **file name** or the '***Folder***' keyword,
- an envelope ***mirenvelope***,
- the **temporal** evolution of a scalar data, such as fluxes in particular (***mirflux***),
- in particular, onset detection curves (***mironsets***) can be decomposed into frames as well.

Syntax

The frame decomposition can be performed using the *mirframe* command. The frames can be specified as follows:

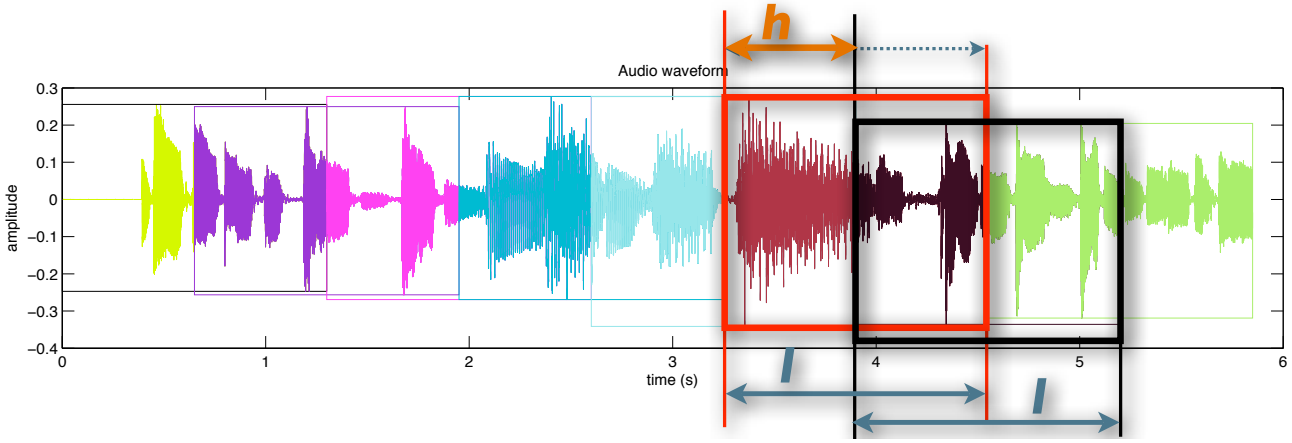
*mirframe(x,..., **'Length', w, wu**):*

- *w* is the length of the window in seconds (default: .05 seconds);
- *u* is the unit, either
  - '***s***' (seconds, default unit),
  - or '***sp***' (number of samples).

*mirframe(x,'..., **'Hop', h, hu**):*

- *h* is the hop factor, or distance between successive frames (default: half overlapping: each frame begins at the middle of the previous frame)
- *u* is the unit, either

- '**/r**' (ratio with respect to the frame length, default unit)
- '**%**' (ratio as percentage)
- '**s**' (seconds)
- '**sp**' (number of samples)
- or '**Hz**' (hertz), i.e., number of frames per second: the exactness of the frame rate is ensured and may cause a slight fluctuation of the elementary hop distances.



Frame decomposition of an audio waveform, with frane length l and hop factor h (represented here, following the default unit, as a ratio with respect to the frame length).

These arguments can also be written as follows (where units can be omitted):

mirframe(x, ~w, wu, h, hu)

CHAINING OF OPERATIONS

Suppose we load an audio file:

a = miraudio('mysong')

then we decompose into frames

f = mirframe(a)

then we can perform any computation on each of the successive frame easily. For instance, the computation of the spectrum in each frame (or spectrogram), can be written as:

s = mirspectrum(f)

The 'Frame' option

The two first previous commands can be condensed into one line, using the 'Frame' option.

*f = miraudio('mysong', **'Frame')***

and the three commands can be condensed into one line also using the 'Frame' option.

*s = mirspectrum('mysong', **'Frame')***

The frame specifications can be expressed in the following way:

*mirspectrum(..., **'Frame', l, 's', h, '/1')***

This '*Frame*' option is available to most operators. Each operator uses specific default values for the '*Frame*' parameters. Each operator can perform the frame decomposition where it is most suitable. For instance, as can be seen in *mironsets* feature map, the 'Frame' option related to the *mironsets* operator will lead to a frame decomposition after the actual computation of the onset detection curve (produced by *mironsets*).

Accessible Output

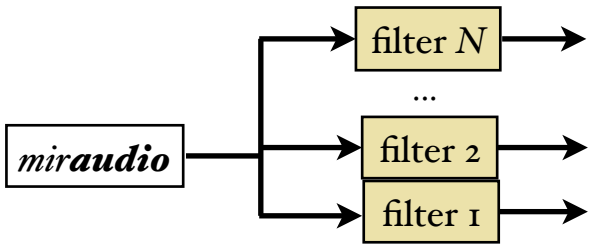
cf. §5.2 for an explanation of the use of the *get* method. Specific fields:

- '***FramePos***': the starting and ending temporal positions of each successive frame, stored in the same way as for '*Data*' (cf. §5.2),
- '***Framed***': whether the data has been decomposed into frames or not.

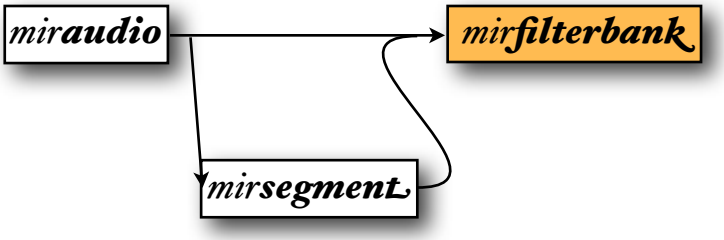
mirfilterbank

Filterbank decomposition

It is often interesting to decompose the audio signal into a series of audio signals of different frequency register, from low frequency channels to high frequency channels. This enables thus to study each of these channels separately. The decomposition is performed by a bank of filters, each one selecting a particular range of frequency values. This transformation models an actual process of human perception, corresponding to the distribution of frequencies into critical bands in the cochlea.



Flowchart Interconnections



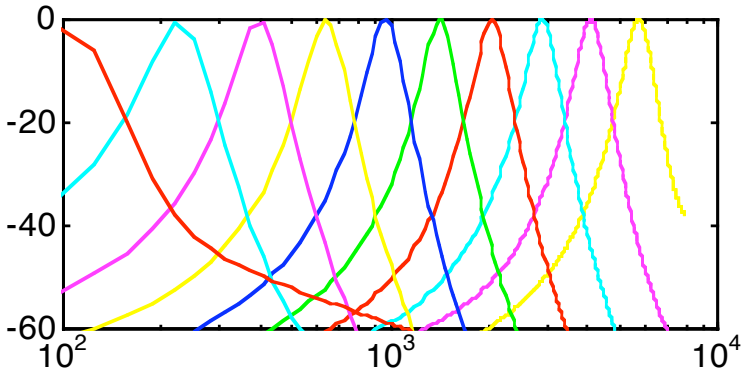
mirfilterbank accepts as input data type either:

- ***miraudio*** objects, where the audio waveform can be segmented (using ***mirsegment***),
- **file name** or the '***Folder***' keyword.

Filterbank Selection

Two basic types of filterbanks are proposed in *MIRtoolbox*:

- *mirfilterbank*(..., '***Gammatone***') carries out a Gammatone filterbank decomposition (Patterson et al, 1992). It is known to simulate well the response of the basilar membrane. It is based on a Equivalent Rectangular Bandwidth (ERB) filterbank, meaning that the width of each band is determined by a particular psychoacoustical law. For Gammatone filterbanks, *mirfilterbank* calls the *Auditory Toolbox* routines *MakeERBFilters* and *ERBfilterbank*. This is the default choice when calling *mirfilterbank*.



Ten ERB filters between 100 and 8000Hz (Slaney, 1998)

- *mirfilterbank(..., '**Lowest**', f)* indicates the lowest frequency *f*, in Hz. Default value: 50 Hz.

- *mirfilterbank(..., '**2Channels**')* performs a computational simplification of the filterbank using just two channels, one for low-frequencies, below 1000 Hz, and one for high-frequencies, over 1000 Hz (Tolonen and Karjalainen, 2000). On the high-frequency channel is performed an envelope extraction using a half-wave rectification and the same low-pass filter used for the low-frequency channel. This filterbank is mainly used for multi-pitch extraction (cf. *mirpitch*).

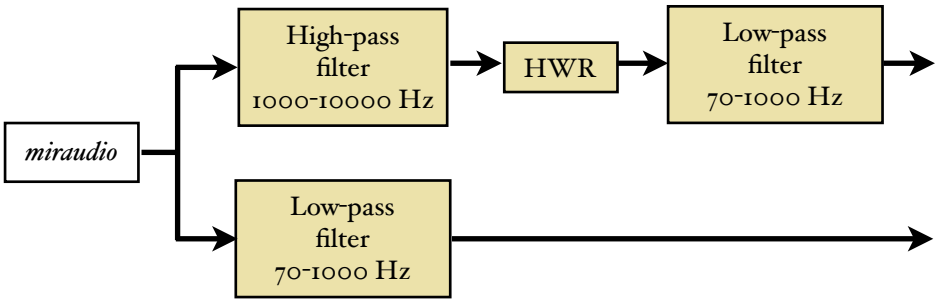


Diagram of the two-channel filterbank proposed in (Tolonen and Karjalainen, 2000)

For these general type of filterbanks are chosen, further options are available:

- *mirfilterbank(..., '**NbChannels**', N)* specifies the number of channels in the bank. By default: *N* = 10. This option is useless for '*2Channels*'.

- *mirfilterbank(..., '**Channel**', c)* – or *mirfilterbank(..., '**Channels**',c)* – only output the channels whose ranks are indicated in the array *c*. (default: *c = (1:N)*)

Manual Specifications

*mirfilterbank(..., '**Manual**', f)* specifies a set of non-overlapping low-pass, band-pass and high-pass eliptic filters (Scheirer, 1998). The series of cut-off frequencies *f* as to be specified as next parameter.

- If this series of frequencies begins with -Inf, the first filter is low-pass.
- If this series of frequencies ends with Inf, the last filter is high-pass.

*mirfilterbank(...,'**Order**', o)* specifies the order of the filters. The default is set to *o* = 4 (Scheirer, 1998)

*mirfilterbank(...,'**Hop**', h)* specifies the degree of spectral overlapping between successive channels.

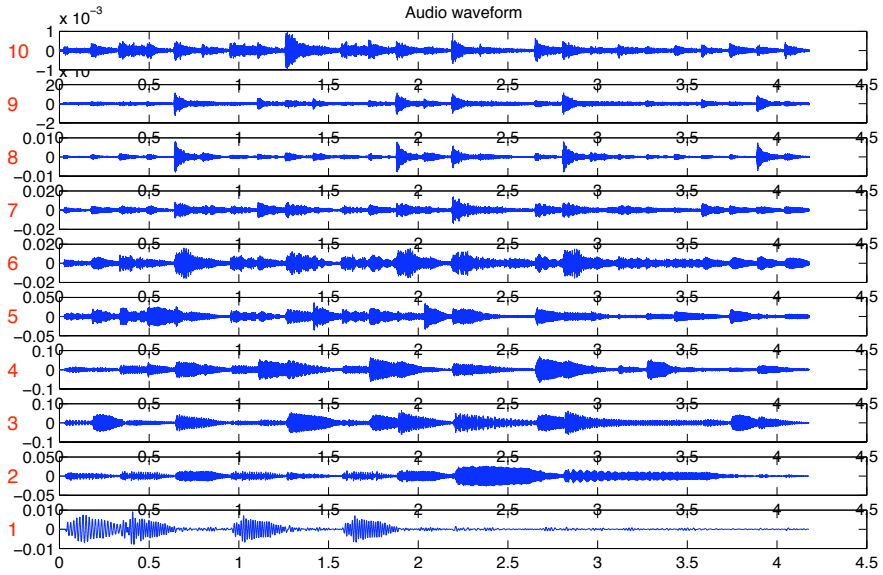
- If *h* = 1 (default value), the filters are non-overlapping.
- If *h* = 2, the filters are half-overlapping.
- If *h* = 3, the spectral hop factor between successive filters is a third of the whole frequency region, etc.

Preselected Filterbanks

mirfilterbank(..., p) specifies predefined filterbanks, all implemented using elliptic filters, by default of order 4:

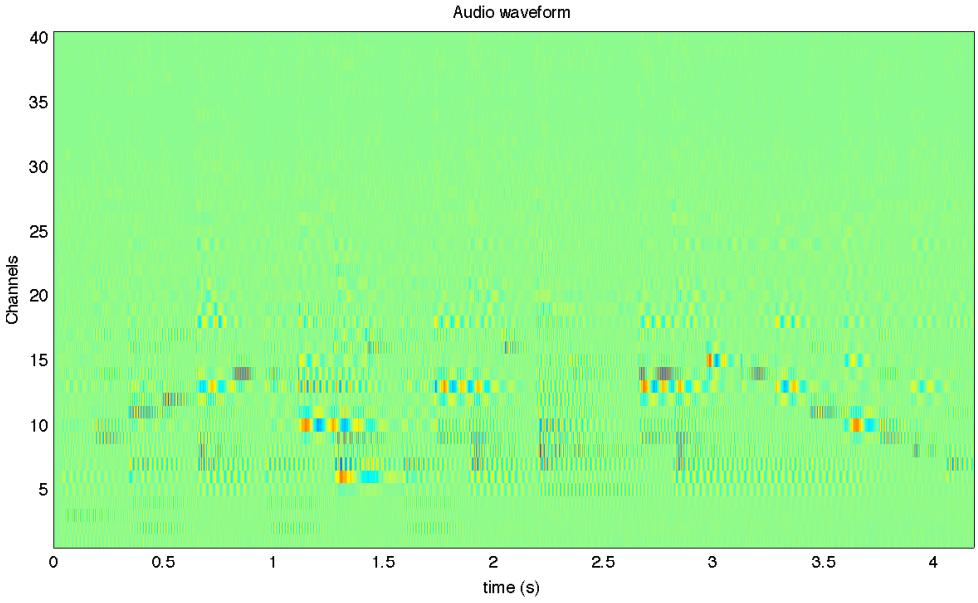
- *p* = *'Mel'*: Mel scale (cf. *mirspectrum(..., 'Mel')*).
- *p* = *'Bark'*: Bark scale (cf. *mirspectrum(..., 'Bark')*).
- *p* = *'Scheirer'* proposed in (Scheirer, 1998) corresponds to *'Manual',[-Inf 200 400 800 1600 3200 Inf]*
- *p* = *'Klapuri'* proposed in (Klapuri, 1999) corresponds to *'Manual',44*[2.^([0:2, (9+(0:17))/3])]*

EXAMPLE



mirfilterbank('ragtime')

If the number of channels exceeds 20, the audio waveform decomposition is represented as a single image bitmap, where each line of pixel represents each successive channel:



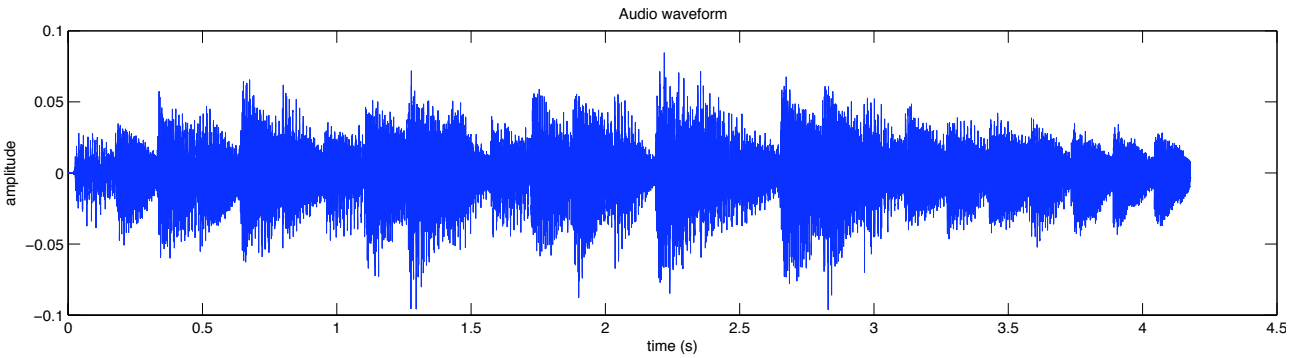
mirfilterbank('ragtime', 'NbChannels', 40)

mirenvelope

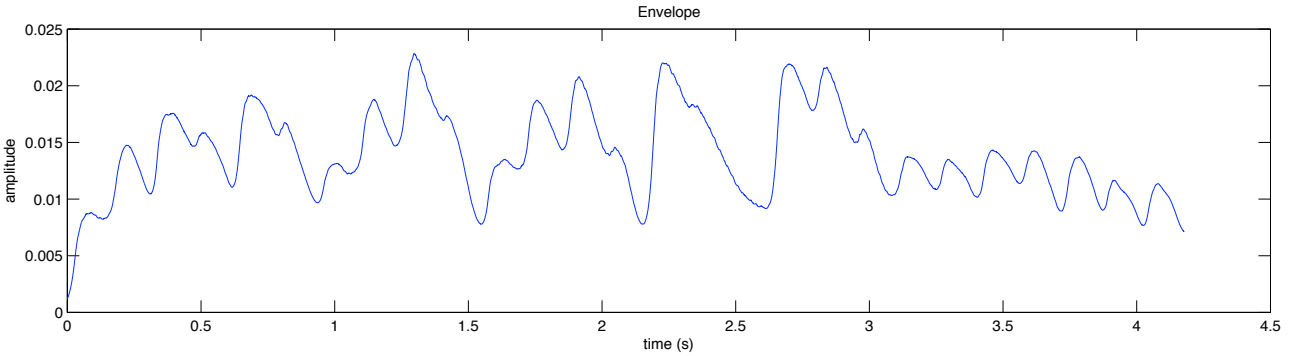
Amplitude envelope

From an audio waveform can be computed the envelope, which shows the global outer shape of the signal. It is particularly useful in order to show the long term evolution of the signal, and has application in particular to the detection of musical events such as notes.

Here is an example of audio file with its envelope:

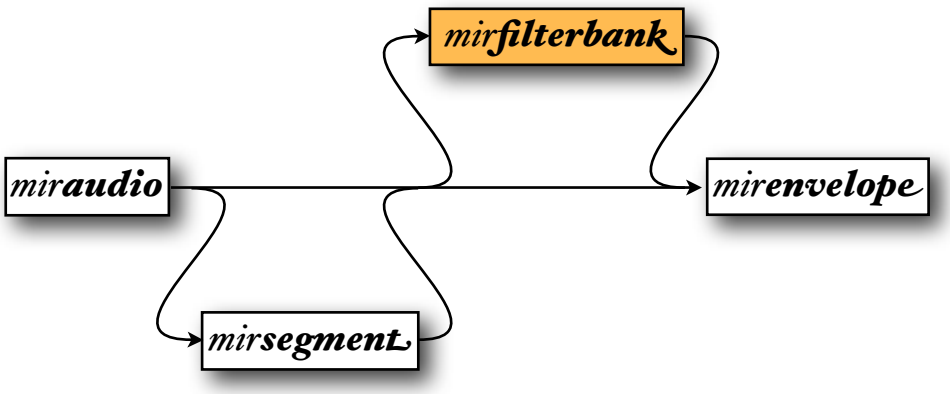


Audio waveform of ragtime excerpt



Corresponding envelope of the ragtime excerpt

Flowchart Interconnections



mirenvelope accepts as input data type either:

- ***miraudio*** objects, where the audio waveform can be segmented (using ***mirsegment***) and/or decomposed into channels (using ***mirfilterbank***),
- **file name** or the '***Folder***' keyword.

Besides, *mirenvelope(..., '**Frame**', ...)* directly performs a frame decomposition on the resulting envelope[1]. Default value: window length of 50 ms and half overlapping.

Parameters specification

The envelope extraction is based on two alternate strategies: either based on a filtering of the signal ('***Filter***' option), or on a decomposition into frames via a spectrogram computation ('***Spectro***' option). Each of these strategies accepts particular options:

- *mirenvelope(...,'**Filter**')* extract the envelope through a filtering of the signal.
  - First the signal can be converted from the real domain to the complex domain using a Hilbert transform. In this way the envelope is estimated in a three-dimensional space defined by the product of the complex domain and the temporal axis. Indeed in this representation the signal looks like a "spring" of varying width, and the envelope would correspond to that varying width. In the real domain, on the other hand, the constant crossing of the signal with the zero axis may sometime give erroneous results.

    An Hilbert transform can be performed in *mirenvelope,* based on the *Matlab* function *hilbert*. In order to toggle on the Hilbert transform, the following keyword should be added:

    *mirenvelope(..., '**Hilbert**')*

    Beware however that, although sometimes the use of the Hilbert transform seems to improve somewhat the results, and might in particular show clearer burst of energy, we noticed some problematic behavior, in particular at the beginning and the end of the signal, and after some particular bursts of energy. This becomes all the more problematic when chunk decompositions are used (cf. §5.3), since the continuity between chunk cannot be ensured any more. For that reason, since version 1.1 of *MIRtoolbox*, the use of Hilbert transform is toggled off by default.

---

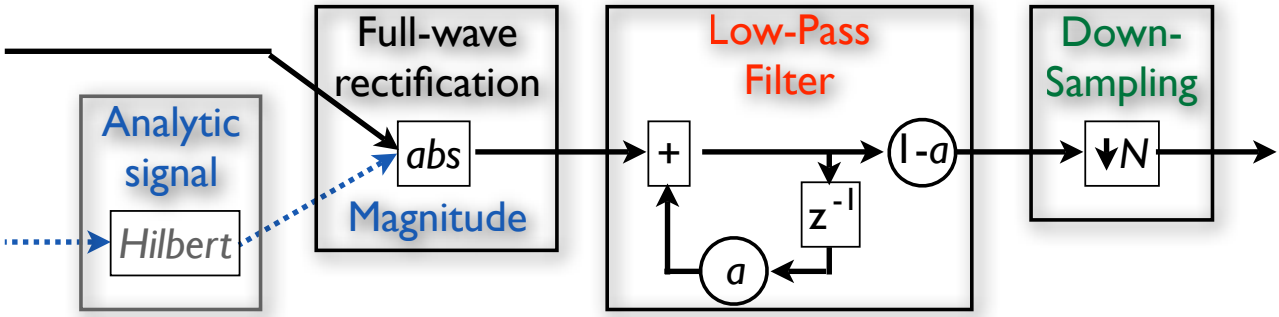
[1] The frame decomposition should not be performed before the envelope extraction, as it would induce significant redundancy in the computation and arouse problems related to the transitory phases at the beginning of each frame.

If the signal is in the real domain, the next step consists in a full-wave rectification, reflecting all the negative lobes of the signal into the positive domain, leading to a series of positive half-wave lobes. The further smoothing of the signal (in the next step) will leads to an estimation of the envelope. If on the contrary the signal is in the complex domain, a direct estimation of the envelope can be obtained by computing the modulus, i.e., the width of the "string". These two operations, either from the real or the complex domains, although apparently different, relate to the same *Matlab* command *abs*.

- *mirenvelope(..., '**PreDecim**', N)* down-samples by a factor $N>1$, where N is an integer, before the low-pass filtering (Klapuri, 1999). Default value: $N = 1$, corresponding to no down-sampling.

- The next step consists in a low-pass filter than retain from the signal only the long-term evolution, by removing all the more rapid oscillations. This is performed through a filtering of the signal. Two types of filters are available, either a simple autoregressive coefficient, with Infinite Impulse Response ('***IIR***' value in '***FilterType***' option), or a half-Hanning (raised cosine) filter ('***HalfHann***' value in '***FilterType***' option).

  - *mirenvelope(..., '**FilterType**', '**IIR**')* extract the envelope using an auto-regressive filter of infinite impulse response (IIR):



Detail of the envelope extraction process

The range of frequencies to be filtered can be controlled by selecting a proper value for the *a* parameter. Another way of expressing this parameter is by considering its time constant. If we feed the filter with a step function (i.e. 0 before time 0, and 1 after time 0), the time constant will correspond to the time it will take for the output to reach 63 % of the input. Hence higher time constant means smoother filtering. The default time constant is set to .02 seconds and can be changed using the option:

*mirenvelope(..., '**Tau**', t)*

Remarks:

1. As low-pass filters actually lead to a shifting of the phases of the signal. This is counteracted using a second filtering of the reverse signal. The time constant *t* is the time constant of each separate filter, therefore *the resulting time constant is around twice bigger*.
2. The reverse filtering is not performed using Matlab *filtfilt* function since version 1.1 of *MIRtoolbox* – because this would not work in the case of chunk decomposition (cf. §5.3) – but has been partly re-implemented. In particular, contrary to *filtfilt*, care is not yet taken to minimize startup and ending transients by matching initial conditions.

- Once the signal has been smoothed, as there is a lot of redundancy between the successive samples, the signal can be down-sampled. The default parameter related to down-sampling is the down-sampling rate *N,* i.e. the *integer* ratio between the old and the new sampling rate. *N* is set by default to 16, and can be changed using the option:

mirenvelope(..., 'PostDecim', N)

Alternatively, any sampling rate *r* (in Hz) can be specified using the post-processing option '*Sampling*'.

- *mirenvelope(..., 'Trim')*: trims the initial ascending phase of the curves related to the transitory state.

- *mirenvelope(..., 'Spectro')* extracts the envelope through the computation of a power spectrogram, with frame size 100 ms, hop factor 10% and the use of Hanning windowing:

mirspectrum(..., 'Frame', .1, 's', .1, '/1', 'Window', 'hanning', 'Power', b)

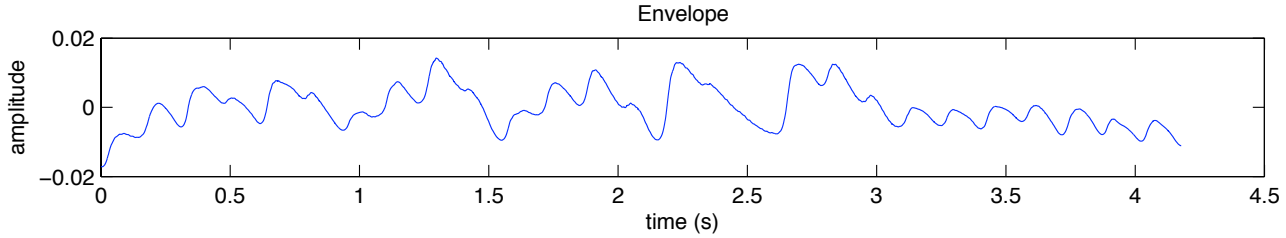
- *mirenvelope(..., b)* specifies whether the frequency range is further decomposed into bands (cf. *mirspectrum*). Possible values:
  - *b* = *'Freq'*: no band decomposition (default value),
  - *b* = *'Mel'*: Mel-band decomposition,
  - *b* = *'Bark'*: Bark-band decomposition,
  - *b* = *'Cents'*: decompositions into cents.
- *mirenvelope(..., 'Frame',...)* modifies the default frame configuration.
- *mirenvelope(..., 'UpSample', N)* upsamples by a factor *N*>1, where *N* is an integer. Default value if '*UpSample*' called: *N* = 2

- *mirenvelope(..., **'Complex'**)* toggles on the '***Complex***' method for the spectral flux computation (cf. *mirflux*).

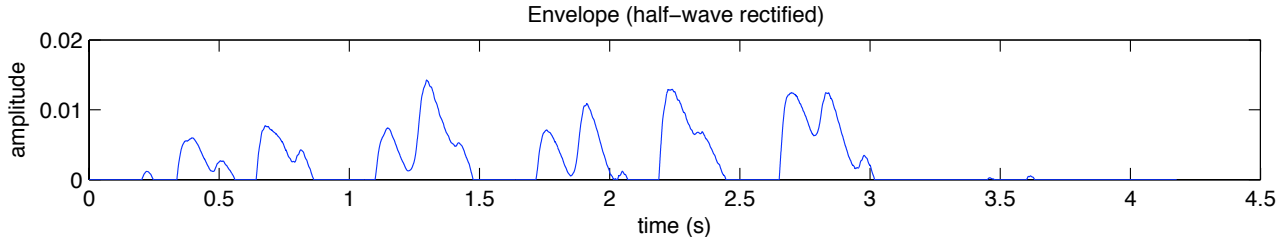
POST-PROCESSING OPTIONS

Different operations can be performed on the envelope curve:

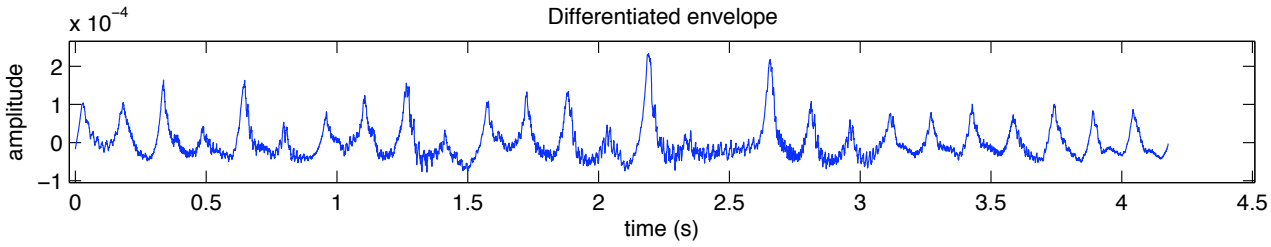
- *mirenvelope(...,**'Sampling'**, r)* resamples to rate *r* (in Hz). '***PostDecim***' and '***Sampling***' options cannot therefore be combined.
- *mirenvelope(...,**'Halfwave'**)* performs a half-wave rectification on the envelope.
- *mirenvelope(...,**'Center'**)* centers the extracted envelope.



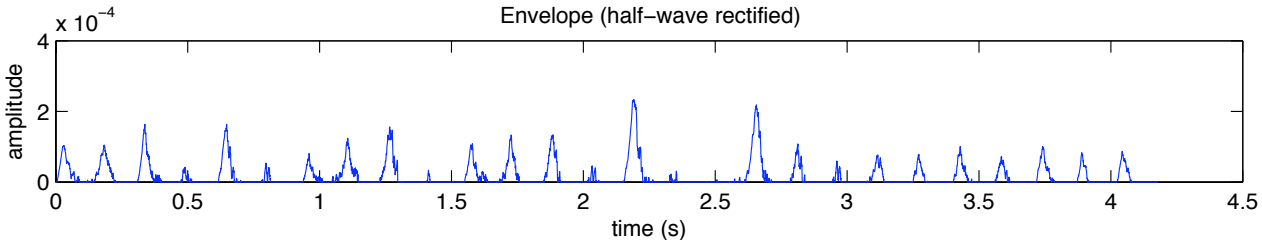
- *mirenvelope(...,**'HalfwaveCenter'**)* performs a half-wave rectification on the centered envelope.



- *mirenvelope(...,**'Log'**)* computes the common logarithm (base 10) of the envelope.
- *mirenvelope(...,**'Mu'**, mu)* computes the logarithm of the envelope, before the eventual differentiation, using a mu-law compression (Klapuri et al., 2006). Default value for *mu*: 100
- *mirenvelope(...,**'Power'**)* computes the power (square) of the envelope.
- *mirenvelope(...,**'Diff'**)* computes the differentiation of the envelope, i.e., the differences between successive samples.



- *mirenvelope(...,'**HalfwaveDiff**')* performs a half-wave rectification on the differentiated envelope.



- *mirenvelope(...,'**Normal**')* normalizes the values of the envelope by fixing the maximum value to 1.
- *mirenvelope(...,'**Lambda**', l)* sums the half-wave rectified envelope with the non-differentiated envelope, using the respective weight 0<l<1 and (1-l). (Klapuri et al., 2006).
- *mirenvelope(...,'**Smooth**',o)* smooths the envelope using a movering average of order *o*. The default value when the option is toggled on: *o*=30
- *mirenvelope(...,'**Gauss**',o)* smooths the envelope using a gaussian of standard deviation *o* samples. The default value when the option is toggled on: *o*=30

Preselected Model

Complete (or nearly complete) model is available:

- *mirenvelope(..., '**Klapuri06**')* follows the model proposed in (Klapuri et al., 2006). Il corresponds to

e = mirenvelope(..., 'Spectro', 'UpSample', 'Mu', 'HalfwaveDiff', 'Lambda', .8);

mirsum(e, 'Adjacent', 10)

Accessible Output

cf. §5.2 for an explanation of the use of the *get* method. Specific fields:

- '***Time***': the temporal positions of samples (same as '*Pos*'),
- '***DownSampling***': the value of the '*PostDecim*' option,
- '***Halfwave***': whether the envelope has been half-wave rectified (1) or not (0),
- '***Diff***': whether the envelope has been differentiated (1) or not (0),
- '***Centered***': whether the envelope is centered (1) or not (0),

• '***Phase***': the phase of the spectrogram, if necessary.

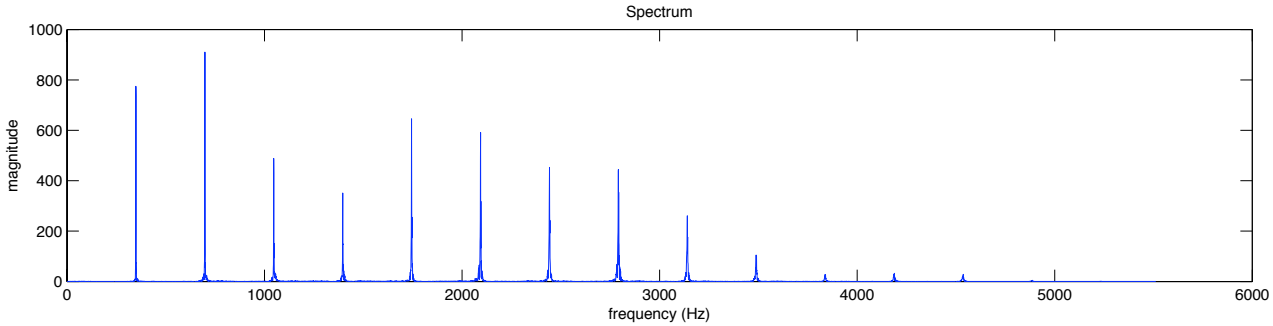
mirspectrum

Fourier transform

A decomposition of the energy of a signal (be it an audio waveform, or an envelope, etc.) along frequencies can be performed using a Discrete Fourier Transform, which, for an audio signal x has for equation:

$$X_k = \sum_{n=0}^{N-1} x_n e^{-\frac{2\pi i}{N} kn} \qquad k = 0, \ldots, N-1$$

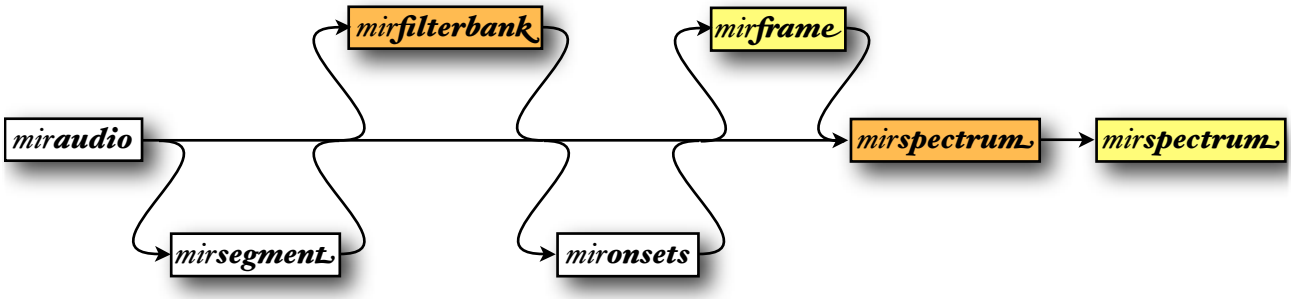
This decomposition is performed using a Fast Fourier Transform by the *mirspectrum* function by calling Matlab *fft* function. The graph returned by the function highlights the repartition of the amplitude of the frequencies (i.e., the modulus of *Xk* for all *k*), such as the following:



We can also obtain for each frequency the actual phase position (i.e., the phase of *Xk*), which indicates the exact position of each frequency component at the instant *t* = 0. If the result of the spectrum decomposition is *s*, the phase spectrum is obtained by using the command:

get(s, 'Phase')

Flowchart Interconnections



mirspectrum accepts as input data type either:

- ***miraudio*** objects, where the audio waveform can be segmented (using ***mirsegment~***), decomposed into channels (using ***mirfilterbank***), and/or decomposed into frames (using ***mirframe~*** or the *'**Frame**'* option, with by default a frame length of 50 ms and half overlapping);
- **file name** or the *'**Folder**'* keyword;
- data in the onset detection curve category (cf. ***mironsets***):
  - ***mirenvelope~*** objects, frame-decomposed or not,
  - fluxes (cf. ***mirflux***), frame-decomposed or not;
- ***mirspectrum~*** frame-decomposed objects: by calling again *mirspectrum~* with the *'**AlongBands**'* option, Fourier transforms are computed this time on each temporal signal related to each separate frequency bin (or frequency band, cf. below).

Parameters specification

The range of frequencies, in Hz, can be specified by the options:

- *mirspectrum(..., '**Min~**', mi)* indicates the lowest frequency taken into consideration, expressed in Hz. Default value: 0 Hz.
- *mirspectrum(..., '**Max**', ma)* indicates the highest frequency taken into consideration, expressed in Hz. Default value: the maximal possible frequency, corresponding to the sampling rate divided by 2.
- *mirspectrum(..., '**Window**', ~w)* specifies the windowing method. Windows are used to avoid the problems due to the discontinuities provoked by finite signals. Indeed, an audio sequence is not infinite, and the application of the Fourier Transform requires to replace the infinite time before and after the sequence by zeroes, leading to possible discontinuities at the borders. Windows are used to counteract those discontinuities. Possible values for *~w* are either *~w* = 0 (no windowing) or any windowing function proposed in the *Signal Processing Toolbox*[2]. Default value: *~w* = '*hamming*', the Hamming window being a particular good window for Fourier Transform.
- *mirspectrum(..., '**NormalInput~**')* normalizes the waveform between 0 and 1 before computing the Fourier Transform.
- *mirspectrum(..., '**Phase**', 'No')* does not compute the related FFT phase. The FFT phase is not computed anyway whenever another option that will make the phase information irrelevant (such as '*Log*', '*dB*', etc.) is specified.

---

[2] The list of possible window arguments can be found in the *~window* documentation (*help window*).

Resolution specification

The frequency resolution of the spectrum directly depends on the size of the audio waveform: the longer the waveform, the better the frequency resolution. It is possible, however, to increase the frequency resolution of a given audio waveform by simply adding a series of zeros at the end of the sequence, which is called *zero-padding*. Besides, an optimized version of the Discrete Fourier Transform, called Fast Fourier Transform (FFT) can be performed if the length of the audio waveform (including the zero-padding) is a power of 2. For this reason, by default, a zero-padding is performed by default in order to ensure that the length of the audio waveform is a power of 2. But these operations can be tuned individually:

- *mirspectrum(...,'**MinRes**', mr)* adds a constraint related to the a minimal frequency resolution, fixed to the value *mr* (in Hz). The audio waveform is automatically zero-padded to the lowest power of 2 ensuring the required frequency resolution.
- *mirspectrum(..., 'MinRes', r, '**OctaveRatio**', tol)*: Indicates the minimal accepted resolution in terms of number of divisions of the octave. Low frequencies are ignored in order to reach the desired resolution. The corresponding required frequency resolution is equal to the difference between the first frequency bins, multiplied by the constraining multiplicative factor *tol* (set by default to .75).
- *mirspectrum(...,'**Res**', r)* specifies the frequency resolution *r* (in Hz) that will be secured as closely as possible, through an automated zero-padding. The length of the resulting audio waveform will not necessarily be a power of 2, therefore the FFT routine will rarely be used.
- *mirspectrum(...,'**Length**', l)* specifies the length of the audio waveform after zero-padding. If the length is not a power of 2, the FFT routine will not be used.
- *mirspectrum(...,'**ZeroPad**', s)* performs a zero-padding of *s* samples. If the total length is not a power of 2, the FFT routine will not be used.
- *mirspectrum(...,'**WarningRes**', mr)* indicates a required frequency resolution, in Hz, for the input signal. If the resolution does not reach that prerequisite, a warning is displayed.

Alternatively, the spectrum decomposition can be performed through a Constant Q Transform instead of a FFT, which enables to express the frequency resolution as a constant number of bins per octave:

- *mirspectrum(...,'**ConstantQ**', nb)* fixes the number of bins per octave to *nb*. Default value when the '*ConstantQ*' option is toggled on: *nb*=12 bins per octave.

Please note however that the Constant Q Transform is implemented as a *Matlab* M file, whereas *Matlab's* FFT algorithm is optimized, therefore faster.

POST-PROCESSING OPTIONS

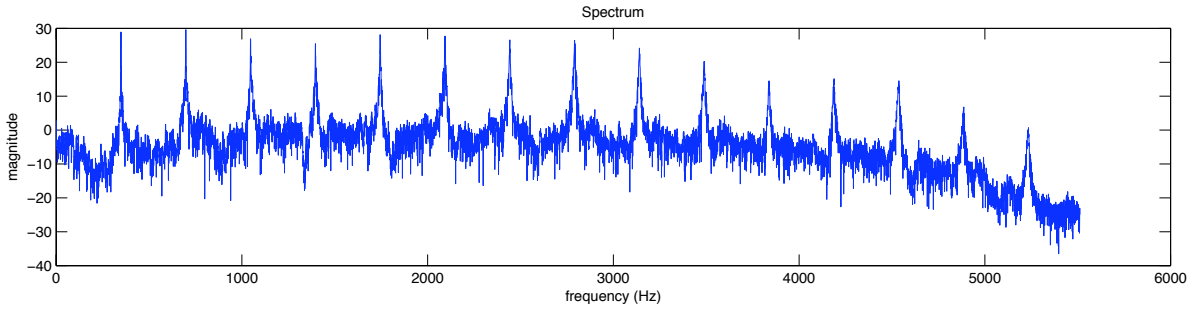
- *mirspectrum(...,**'Terhardt'**)* modulates the energy following (Terhardt, 1979) outer ear model. The function is mainly characterized by an attenuation in the lower and higher registers of the spectrum, and an emphasis around 2–5 KHz, where much of the speech information is carried. (Code based on Pampalk's MA toolbox).

- *mirspectrum(..., **'Normal'**)* normalizes with respect to energy: each magnitude is divided by the euclidian norm (root sum of the squared magnitude).

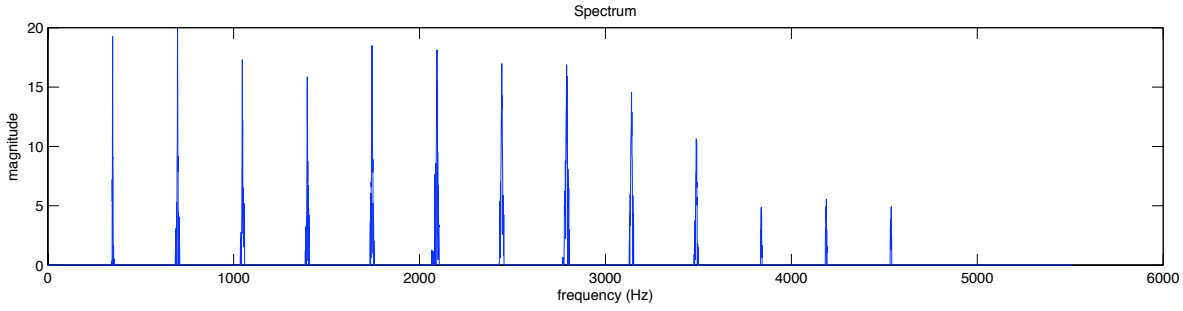
- *mirspectrum(..., **'NormalLength'**)* normalizes with respect to the duration (in s.) of the audio input data.

- *mirspectrum(...,**'Power'**)* squares the energy: each magnitude is squared.

- *mirspectrum(..., **'dB'**)* represents the spectrum energy in decibel scale. For the previous example we obtain the following spectrum:



- *mirspectrum(..., 'dB', th)* keeps only the highest energy over a range of *th* dB. For example if we take only the 20 most highest dB in the previous example we obtain:



- *mirspectrum(...,**'Resonance'**, r)* multiplies the spectrum curve with a resonance curve that emphasizes pulsations that are more easily perceived. Two resonance curves are available:

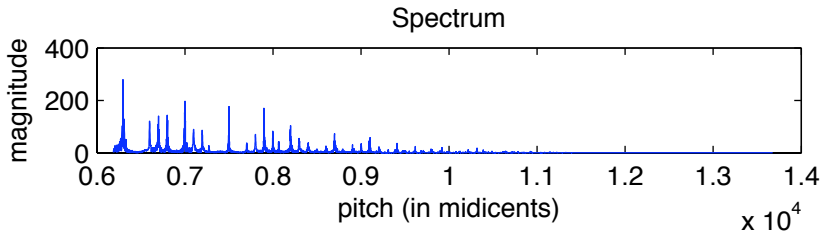
  - *r* = ***'ToiviainenSnyder'*** (Toiviainen & Snyder 2003), default choice, used for onset detection (cf. *mirtempo*),

  - *r* = ***'Fluctuation'***: fluctuation strength (Fastl 1982), default choice for frame-decomposed *mirspectrum* objects redecomposed in '*Mel*' bands (cf. *mirfluctuation*).

- *mirspectrum(...,'**Smooth**', o)* smooths the envelope using a movering average of order *o*. Default value when the option is toggled on: *o*=10
- *mirspectrum(...,'**Gauss**', o)* smooths the envelope using a gaussian of standard deviation *o* samples. Default value when the option is toggled on: *o*=10

FREQUENCY REDISTRIBUTION

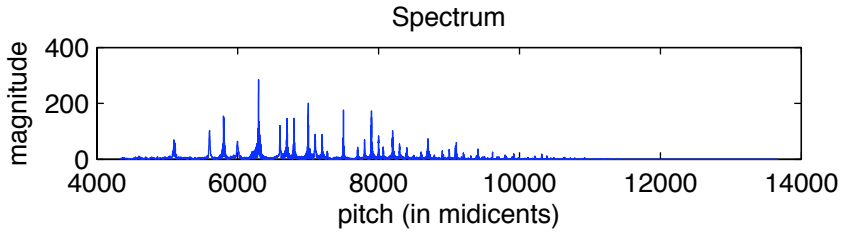
- *mirspectrum(..., **'Cents'**)* redistributes the frequencies along cents. Each octave is decomposed into 1200 bins equally distant in the logarithmic representation. The frequency axis is hence expressed in MIDI-cents unit: to each pitch of the equal temperament is associated the corresponding MIDI pitch standard value multiply by 100 (69*100=6900 for A4=440Hz, 70*100=7000 for B4, etc.).



mirspectrum('ragtime','Cents')

It has to be noticed that this decomposition requires a frequency resolution that gets higher for lower frequencies: a cent-distribution starting from infinitely low frequency (near 0 Hz would require an infinite frequency resolution). Hence by default, the cent-decomposition is defined only for the frequency range suitable for the frequency resolution initially associated to the given spectrum representation. Two levers are available here:

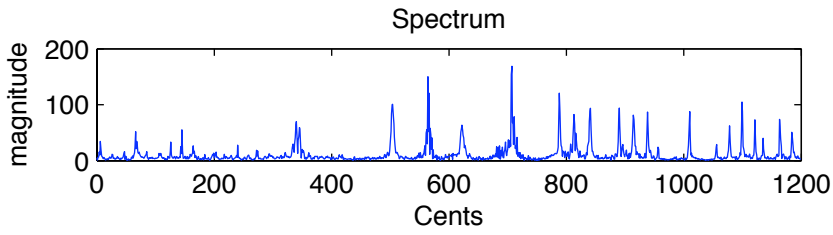
- If a minimal frequency range for the spectrum representation has been set (using the *'Min'* parameter), the frequency resolution of the spectrum is automatically set in order to meet that particular requirement.



mirspectrum('ragtime','Cents','Min',100)

- By increasing the frequency resolution of the spectrum (for instance by using the *'Res'* or *'MinRes'* parameters), the frequency range will be increased accordingly.

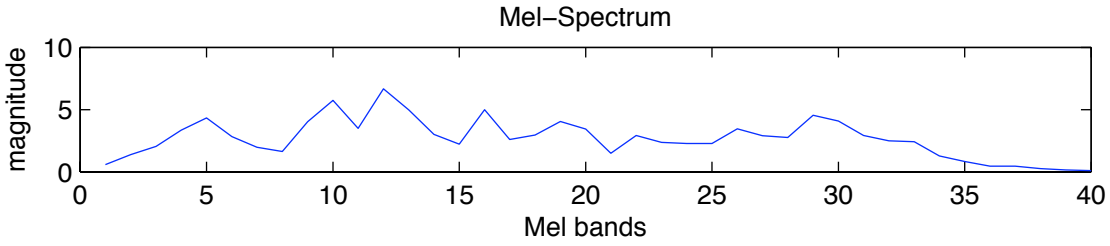
- *mirspectrum(..., **'Collapsed'**)* collapses the cent-spectrum into one octave. In the resulting spectrum, the abscissa contains in total 1200 bins, representing the 1200 cents of one octave, and each bin contains the energy related to one position of one octave and of all the multiple of this octave.



mirspectrum('ragtime','Cents','Min',100,'Collapsed')

- *mirspectrum(..., **'Mel'**)* redistributes the frequencies along Mel bands. The Mel-scale of auditory pitch was established on the basis of listening experiments with simple tones (Stevens and Volkman, 1940). The Mel scale is now mainly used for the reason of its historical priority only. It is closely related to the Bark scale. It requires the *Auditory Toolbox*.
  - *mirspectrum(..., **'Bands'**, b)* specifies the number of band in the decomposition. By default *b* = 40.

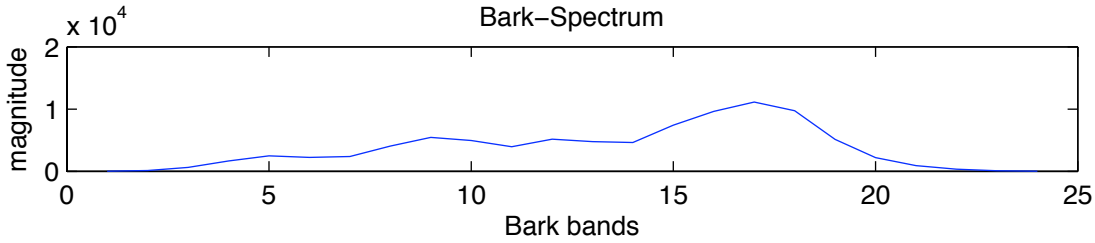
In our example we obtain the following:



The Mel-scale transformation requires a sufficient frequency resolution of the spectrum: as the lower bands are separated with a distance of 66 Hz, the frequency resolution should be higher than 66 Hz in order to ensure that each Mel band can be associated with at least one frequency bin of the spectrum. If the '*Mel*' option is performed in the same *mirspectrum* command that performs the actual FFT, then the minimal frequency resolution is implicitly ensured, by forcing the minimal frequency resolution ('*MinRes*' parameter) to be equal or below 66 Hz. If on the contrary the '*Mel*' is performed in a second step, and if the frequency resolution is worse than 66 Hz, then a warning message is displayed in the Command Window.

- *mirspectrum(..., **'Bark'**)* redistributes the frequencies along critical band rates (in Bark). Measurement of the classical "critical bandwidth" typically involves loudness summation experiments (Zwicker et al., 1957). The critical band rate scale differs from Mel-scale mainly in that it uses the critical band as a natural scale unit. The code is based on the *MA* toolbox.

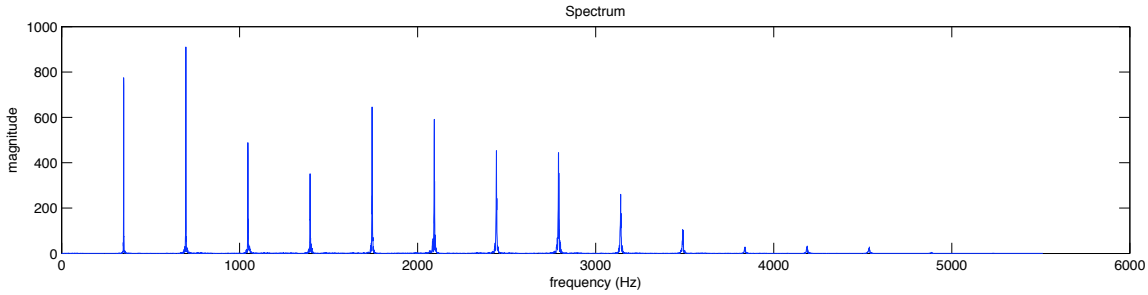
- *mirspectrum(..., 'Mask')* models masking phenomena in each band: when a certain energy appears at a given frequency, lower frequencies in the same frequency region may be unheard, following particular equations. By modeling these masking effects, the unheard periodicities are removed from the spectrum. The code is based on the *MA* toolbox. In our example this will lead to:



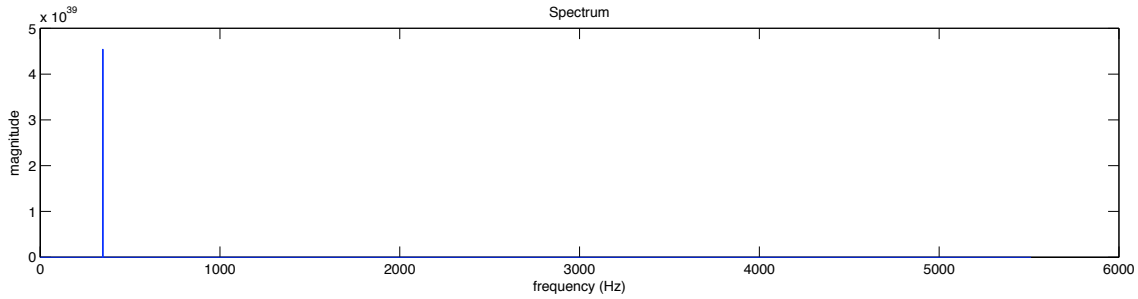
HARMONIC SPECTRAL ANALYSIS

A lot of natural sounds, especially musical ones, are harmonic: each sound consists of a series of frequencies at a multiple ratio of the one of lowest frequency, called fundamental. Techniques have been developed in signal processing to reduce each harmonic series to its fundamental, in order to simplify the representation. *MIRtoolbox* includes two related techniques for the attenuation of harmonics in spectral representation (Alonso et al, 2003):

- *mirspectrum(..., 'Prod', m)* Enhances components that have harmonics located at multiples of range(s) m of the signal's fundamental frequency. Computed by compressing the signal by thea list of factors m, and by multiplying all the results with the original signal. Default value is m = 1:6. Hence for this initial spectrum:



we obtain this reduced spectrum:



- *mirspectrum(..., '**Sum**', m)* Similar idea using addition of the multiples instead of multiplication.

Accessible Output

cf. §5.2 for an explanation of the use of the *get* method. Specific fields:

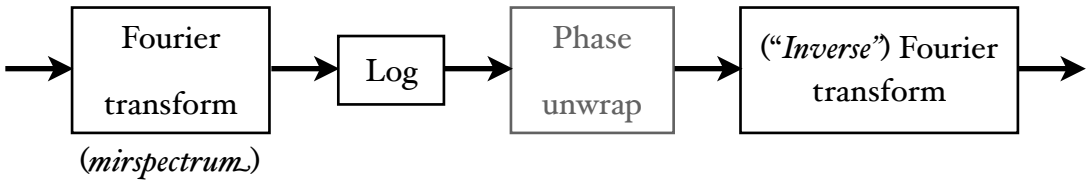
- '***Frequency***': the frequency (in Hz.) associated to each bin (same as '*Pos*'),
- '***Magnitude***': the magnitude associated to each bin (same as '*Data*'),
- '***Phase***': the phase associated to each bin,
- '***XScale***': whether the frequency scale has been redistributed into cents – with ('*Cents(Collapsed)*') or without ('*Cents*') collapsing into one octave –, mels ('*Mel*'), barks ('*Bark*'), or not redistributed at all ('*Freq*'),
- '***Power***': whether the spectrum has been squared (1) or not (0),
- '***Log***': whether the spectrum is in log-scale (1) or in linear scale (0).

mircepstrum

Spectral analysis of spectrum

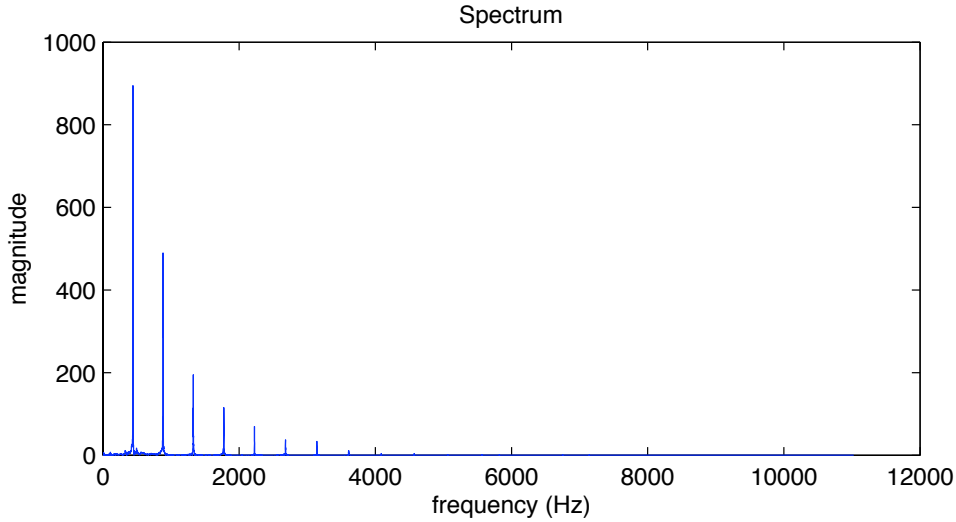
The harmonic sequence can also be used for the detection of the fundamental frequency itself. One idea is to look at the spectrum representation, and try to automatically detect these periodic sequences. And one simple idea consists in performing a Fourier Transform of the Fourier Transform itself, leading to a so-called cepstrum (Bogert et al., 1963).

So if we take the complex spectrum (X_k in the equation defining *mirspectrum*), we can operate the following chain of operations:

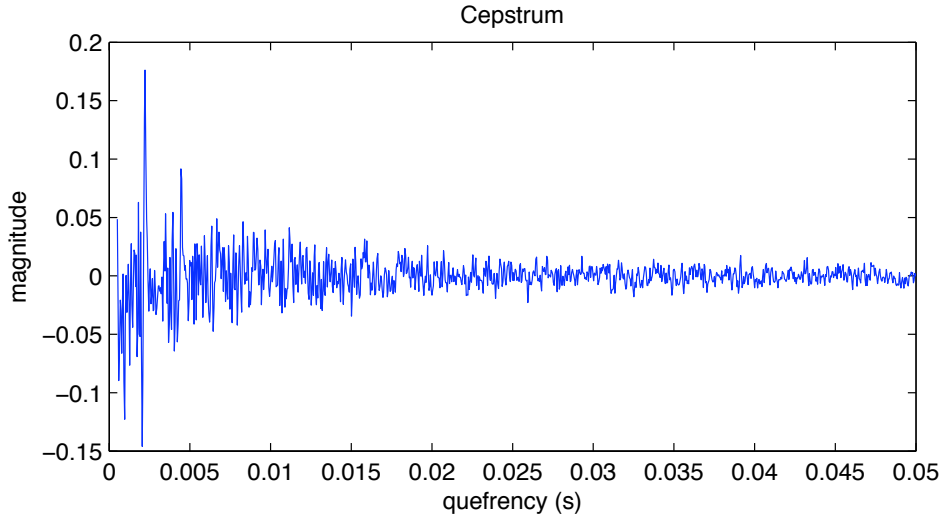


- First a logarithm is performed in order to allow an additive separability of product components of the original spectrum. For instance, for the voice in particular, the spectrum is composed of a product of a vocal cord elementary burst, their echoes, and the vocal track. In the logarithm representations, these components are now added one to each other, and we will then be able to detect the periodic signal as one of the components.
- Then because the logarithm provokes some modification of the phase, it is important to ensure that the phase remains continuous.
- Finally the second Fourier transform is performed in order to find the periodic sequences. As it is sometime a little difficult to conceive what a Fourier transform of Fourier transform is really about, we can simply say, as most say, that it is in fact an Inverse Fourier Transform (as it is the same thing, after all), and the results can then be expressed in a kind of temporal domain, with unit called "quefrency".

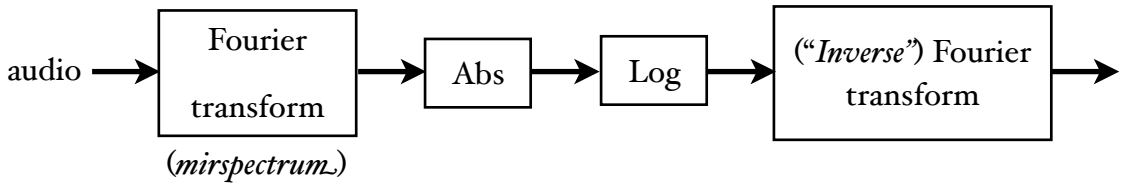
For instance for this spectrum:



we obtain the following cepstrum:

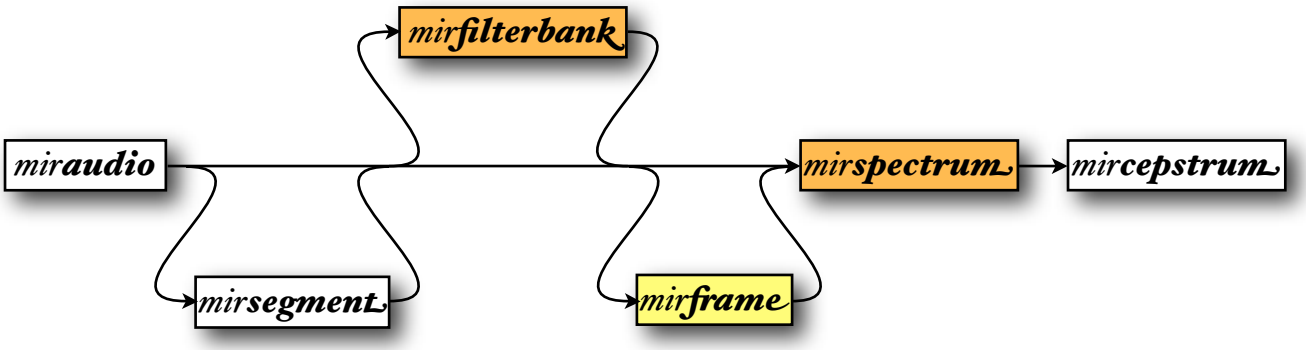


The cepstrum can also be computed from the spectrum amplitude only, by simply taking the logarithm, and directly computing the Fourier transform.



In this case, the phase of the spectrum is not computed.

Flowchart Interconnections



mircepstrum accepts either:

- ***mirspectrum*** objects, or
- ***miraudio*** objects (same as for *mirspectrum*),
- **file name** or the '***Folder***' keyword.

Parameter specifications

- *mircepstrum(..., '**Freq**')*: The results can be represented, instead of using the quefrency domain (in seconds), back to the frequency domain (in Hz) by taking the inverse of each abscissae value. In this frequency representation, each peak is located on a position that directly in dicates the associated fundamental frequency.
- *mircepstrum(..., '**Min**', min)* specifies the lowest delay taken into consideration, in seconds. Default value: 0.0002 s (corresponding to a maximum frequency of 5 kHz). This default value is not set to 0 s in order to exclude the very high peaks confined in the lowest quefrency region: these high peaks seem to come from the fact that the spectrum is a non-centered signal, thus with high (quasi-)stationary energy. However, the value can be forced to 0 using this '*Min*' option.
- *mircepstrum(..., '**Max**', max)* specifies the highest delay taken into consideration, in seconds. Default value: 0.05 s (corresponding to a minimum frequency of 20 Hz). This default value is not set to *Inf* in order to exclude the very high peaks confined in the highest quefrency region: these high peaks seem to come from the fact that the spectrum is a highly variable signal, thus with high energy on its highest frequencies. However, the value can be forced to Inf using this '*Max*' option.
- *mircepstrum(..., '**Complex**')* computes the cepstrum using the complex spectrum. By default, the cepstrum is computed from the spectrum amplitude only.

Accessible Output

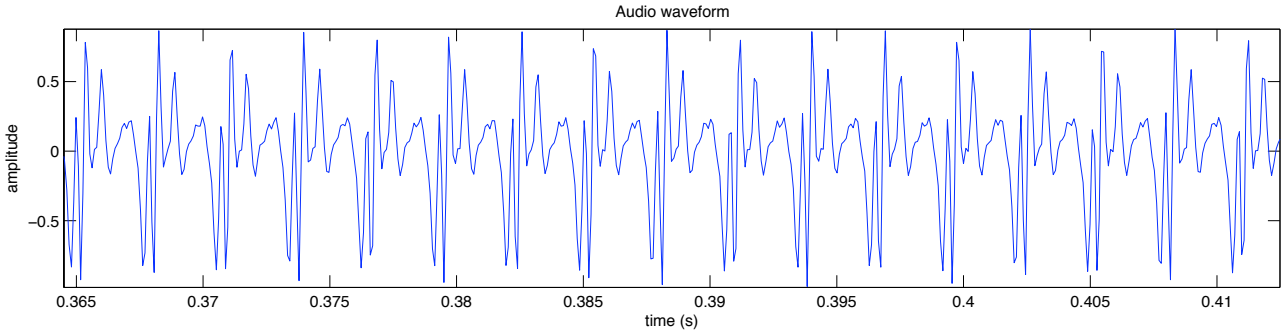
cf. §5.2 for an explanation of the use of the *get* method. Specific fields:

- '***Magnitude***': same as '*Data*',
- '***Phase***': the phase related to the magnitude,
- '***Quefrency***': same as '*Pos*',
- '***FreqDomain***': whether the quefrency are in s. (0) or in Hz. (1).

mirautocor

AUTOCORRELATION FUNCTION

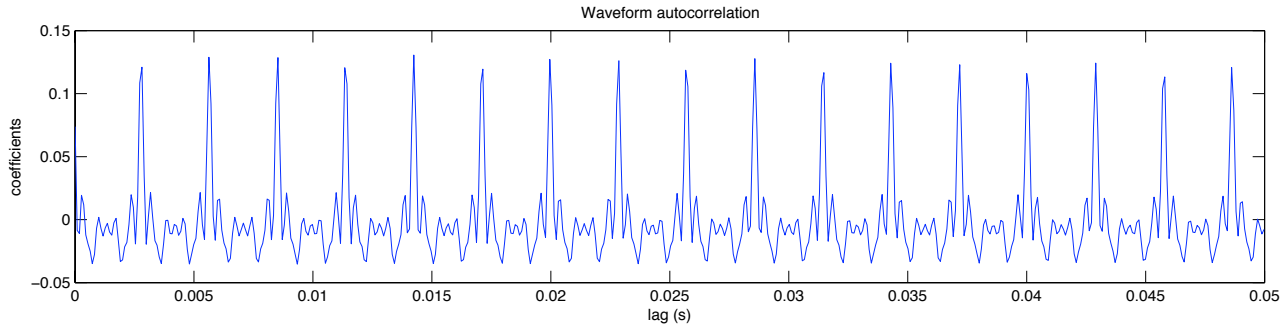
Another way to evaluate periodicities in signals (be it an audio waveform, a spectrum, an envelope, etc.) consists in looking at local correlation between samples. If we take a signal x, such as for instance this trumpet sound:



the autocorrelation function is computed as follows:

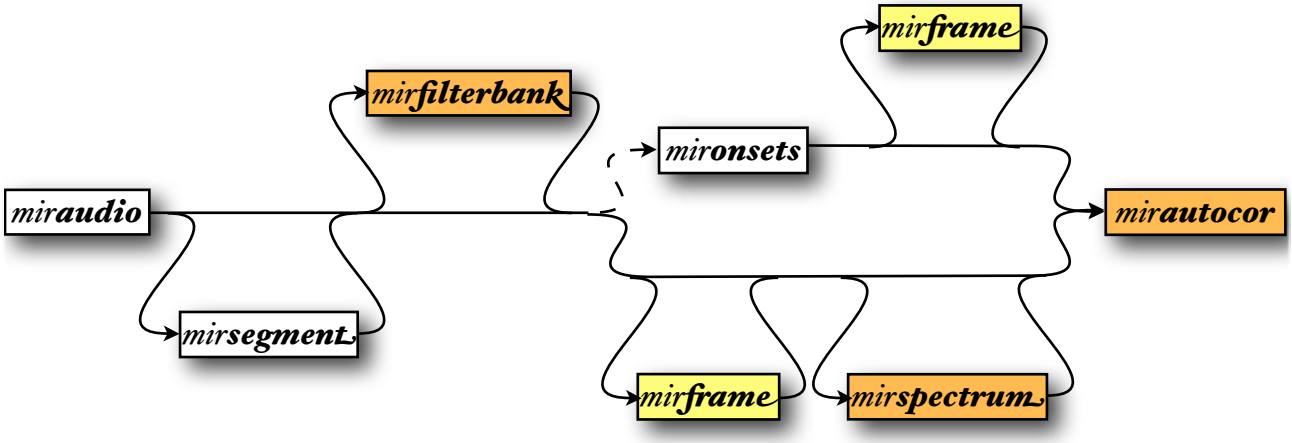
$$R_{xx}(j) = \sum_n x_n \overline{x}_{n-j} \ .$$

For a given lag j, the autocorrelation *Rxx(j) is* computed by multiplying point par point the signal with a shifted version of it of *j* samples. We obtain this curve:



Hence when the lag *j* corresponds to a period of the signal, the signal is shifted to one period ahead, and therefore is exactly superposed to the original signal. Hence the summation gives very high value, as the two signals are highly correlated.

Flowchart Interconnections



mirautocor usually accepts either:

- **file name** or the *'Folder'* keyword,
- ***miraudio*** objects, where the audio waveform can be segmented (using ***mirsegment***), decomposed into channels (using ***mirfilterbank***), and/or decomposed into frames (using ***mirframe*** or the *'Frame'* option, with by default a frame length of 50 ms and half overlapping),
- ***mirspectrum*** objects,
- data in the onset detection curve category (cf. ***mironsets***):
  - ***mirenvelope*** objects, frame-decomposed or not,
  - fluxes (cf. ***mirflux***), frame-decomposed or not,
- ***mirautocor*** objects, for further processing.

Parameters specification

- *mirautocor(..., 'Min', mi)* indicates the lowest delay taken into consideration. Default value: 0 s. The unit can be specified:
  - *mirautocor(..., 'Min', mi, 's')* (default unit)
  - *mirautocor(..., 'Min', mi, 'Hz')*
- *mirautocor(..., 'Max', ma)* indicates the highest delay taken into consideration. The unit can be specified as for '*Min*'. Default value:
  - if the input is an audio waveform, the highest delay is 0.05 s (corresponding to a minimum frequency of 20 Hz).
  - if the input is an envelope, the highest delay is 2 s.

- *mirautocor(..., 'Normal', n)* specifies a normalization option for the cross-correlation (*'biased'*, *'unbiased'*, *'coeff'*, *'none'*). This corresponds exactly to the normalization options in *Matlab xcorr* function, as *mirautocor* actually calls *xcorr* for the actual computation. The default value is *'coeff'*, corresponding to a normalization so that the autocorrelation at zero lag is identically 1. If the data is multi-channel, the normalization is such that the sum over channels at zero lag becomes identically 1. Note however that the *'coeff'* routine is not used when the compression (*'Compres'*) factor *k* is not equal to 2 (see below).

POST-PROCESSING OPTIONS

- *mirautocor(...,'Freq')* represents the autocorrelation function in the frequency domain: the periods are expressed in Hz instead of seconds (see the last curve in the figure below for an illustration).

- *mirautocor(..., 'NormalWindow')* divides the autocorrelation by the autocorrelation of the window. Boersma (1993) shows that by default the autocorrelation function gives higher coefficients for small lags, since the summation is done on more samples. Thus by dividing by the autocorrelation of the window, we normalize all coefficients in such a way that this default is completely resolved. At first sight, the window should simply be a simple rectangular window. But Boersma (1993) shows that it is better to use '*hanning*' window in particular, in order to obtain better harmonic to noise ratio.

  - *mirautocor(..., 'NormalWindow', w)* specifies the window to be used, which can be any window available in the *Signal Processing Toolbox*. Besides w = '*rectangular*' will not perform any particular windowing (corresponding to a rectangular ("invisible") window), but the normalization of the autocorrelation by the autocorrelation of the invisible window will be performed nonetheless. The default value is *w* = '*hanning*'.

  - *mirautocor(..., 'NormalWindow', 'off')* toggles off this normalization (which is '*on*' by default).

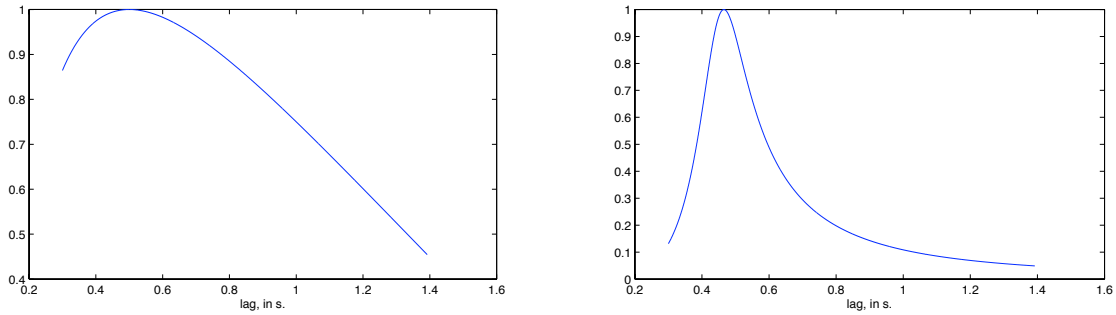
- *mirautocor(..., 'Resonance', r)* multiplies the autocorrelation curve with a resonance curve that emphasizes pulsations that are more easily perceived. Two resonance curves are proposed:

  - *r* = *'ToiviainenSnyder'* (Toiviainen & Snyder 2003) (default value if '*Resonance*' option toggled on),

  - *r* = *'vanNoorden'* (van Noorden & Moelants, 1999).

This option should be used only when the input of the *mirautocor* function is an amplitude envelope, i.e., a *mirenvelope* object.

  - *mirautocor(..., 'Center', c)* assigns the center value of the resonance curve, in seconds. Works mainly with 'ToiviainenSnyder' option. Default value: *c* = 0.5.



Resonance curves 'ToiviainenSnyder' (left) and 'vanNoorden' (right)

- *mirautocor(..., '**Halfwave**')* performs a half-wave rectification on the result, in order to just show the positive autocorrelation coefficients.

GENERALIZED AUTOCORRELATION

*mirautocor(..., '**Compres**', k)* – or equivalently *mirautocor(..., '**Generalized**', k)* – computes the autocorrelation in the frequency domain and includes a magnitude compression of the spectral representation. Indeed an autocorrelation can be expressed using Discrete Fourier Transform as

$$y = IDFT(|DFT(x)|^2),$$

which can be generalized as:

$$y = IDFT(|DFT(x)|^k),$$

Compression of the autocorrelation (i.e., setting a value of k lower than 2) are recommended in (Tolonen & Karjalainen, 2000) because this decreases the width of the peaks in the autocorrelation curve, at the risk however of increasing the sensitivity to noise. According to this study, a good compromise seems to be achieved using value $k = .67$. By default, no compression is performed (hence $k = 2$), whereas if the *'Compress'* keyword is used, value $k = .67$ is set by default if no other value is indicated.

ENHANCED AUTOCORRELATION

In the autocorrelation function, for each periodicity in the signal, peaks will be shown not only at the lag corresponding to that periodicity, but also to all the multiples of that periodicity. In order to avoid such redundancy of information, techniques have been proposed that automatically remove these harmonics. In the frequency domain, this corresponds to sub-harmonics of the peaks.

*mirautocor(..., **'Enhanced'**, a)*: The original autocorrelation function is half-wave rectified, time-scaled by factor a (which can be a factor list as well), and subtracted from the original clipped function (Tolonen & Karjalainen, 2000). If the 'Enhanced' option is not followed by any value, the default value is *a* = 2:10, i.e., from 2 to 10.

If the curve does not start from zero at low lags but begins instead with strictly positive values, the initial discontinuity would be propagated throughout all the scaled version of the curve. In order to avoid this phenomenon, the curve is modified in two successive ways:

- if the curve starts with a descending slope, the whole part before the first local minimum is removed from the curve,
- if the curve starts with an ascending slope, the curve is prolonged to the left following the same slope but which is increased by a factor of 1.1 at each successive bin, until the curve reaches the *x*-axis.

See the figure below for an example of enhanced autocorrelation when computing the pitch content of a piano *Amin3* chord, with the successive step of the default enhancement, as used by default in *mirpitch* (cf. description of *mirpitch*).

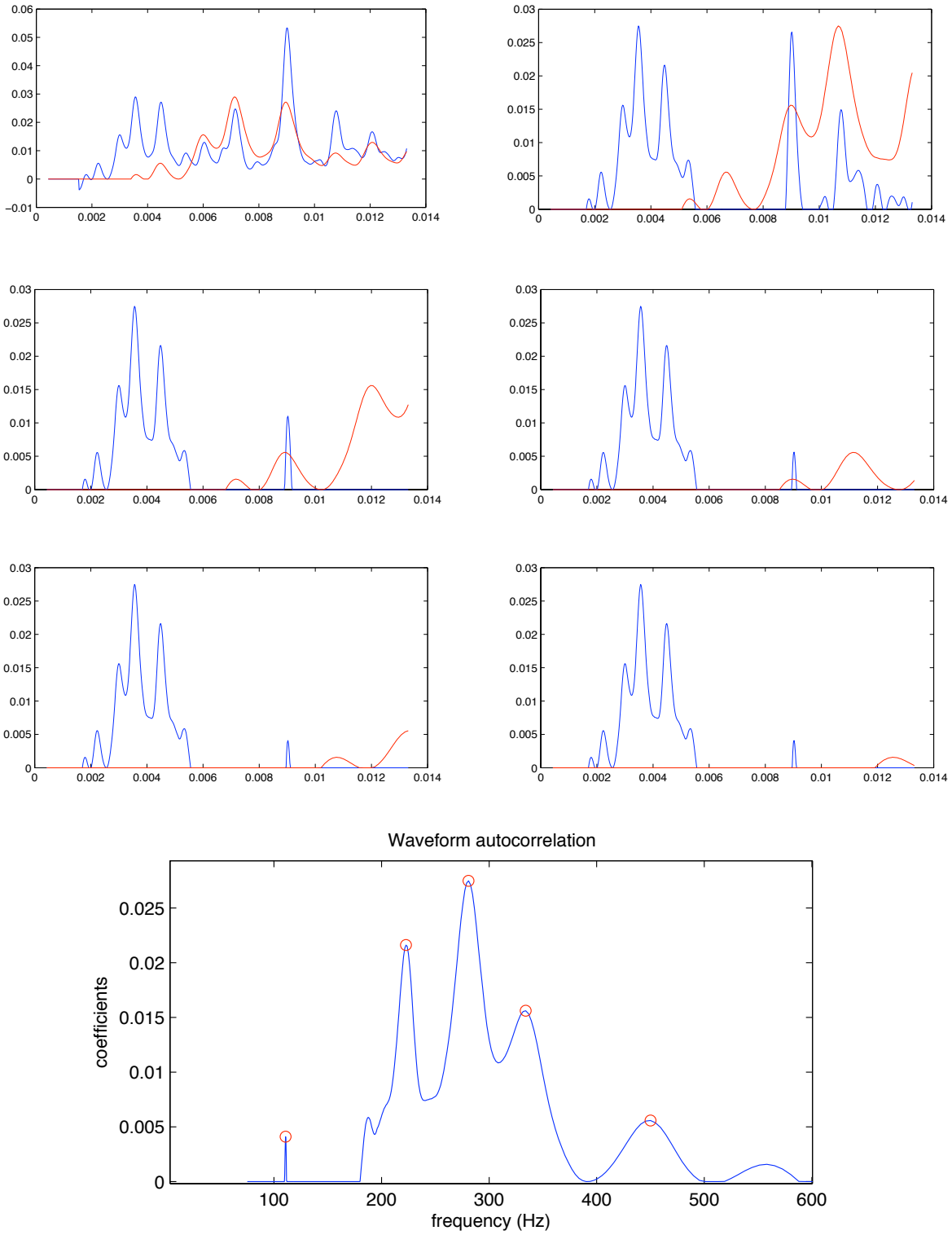


fig 1: Waveform autocorrelation of a piano chord Amaj3 (blue), and scaled autocorrelation of factor 2 (red); fig 2: subtraction of the autocorrelation by the previous scaled autocorrelation (blue), scaled autocorrelation of factor 3 (red); fig 3: resulting subtraction (blue), scaled autocorrelation of factor 4 (red); fig 4: idem for factor 5; fig 5: idem for factor 6; fig 6:idem for factor 7; fig 7: resulting autocorrelation curve in the frequency domain and peak picking

Accessible Output

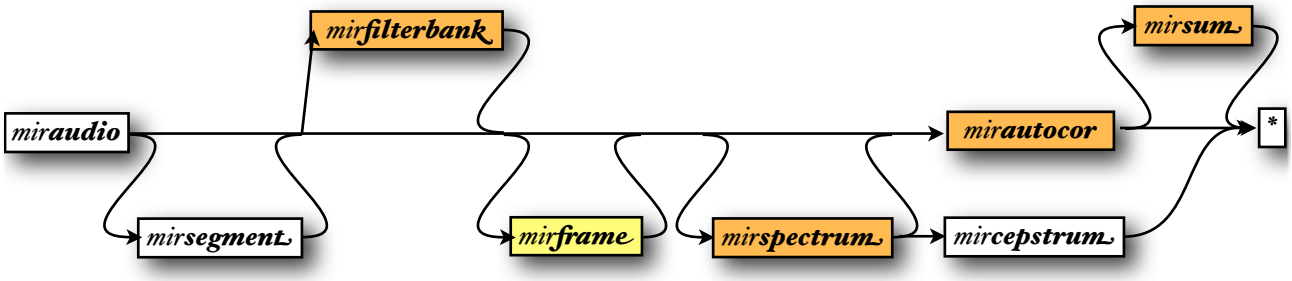
cf. §5.2 for an explanation of the use of the *get* method. Specific fields:

- '***Coeff***': the autocorrelation coefficients (same as '*Data*'),
- '***Lag***': the lags (same as '*Pos*'),
- '***FreqDomain***': whether the lags are in s. (0) or in Hz. (1),
- '***OfSpectrum***': whether the input is a temporal signal (0), or a spectrum (1),
- '***Window***': contains the complete envelope signal used for the windowing.

*

Combining representations

It is also possible to multiple points by points diverse spectral representations and autocorrelation functions, the latter being automatically translated to the spectrum domain (Peeters, 2006). Curves are half-wave rectified before multiplication.

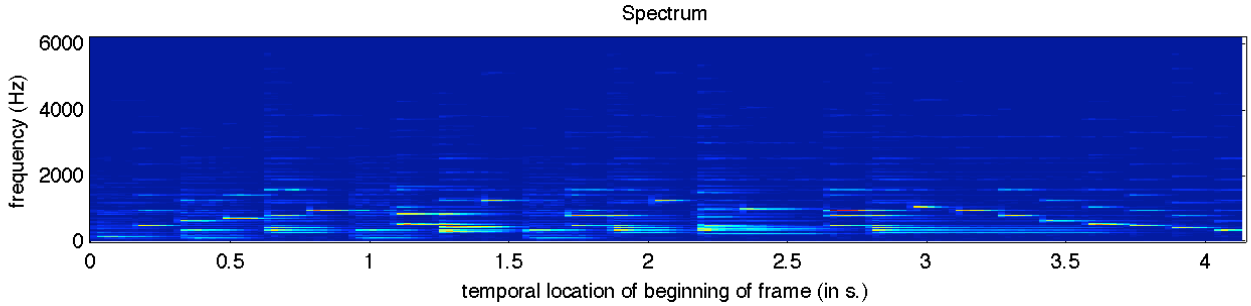


mirflux

DISTANCE BETWEEN SUCCESSIVE FRAMES

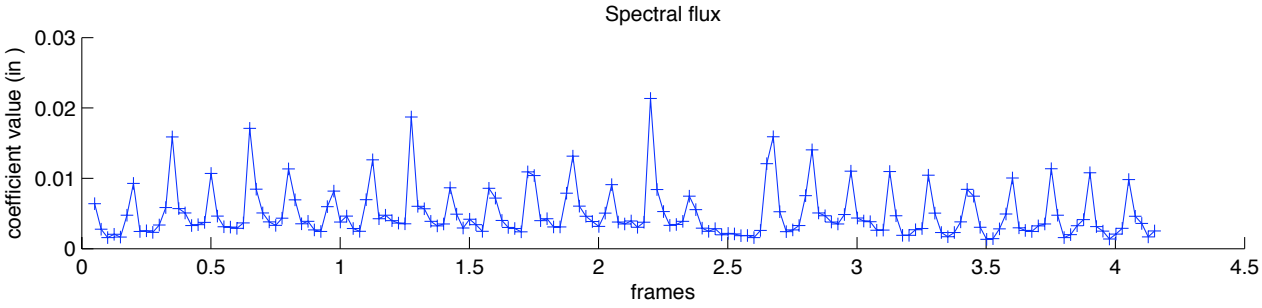
Given a spectrogram:

s = mirspectrum(a, 'Frame')



we can compute the spectral flux as being the distance between the spectrum of each successive frames.

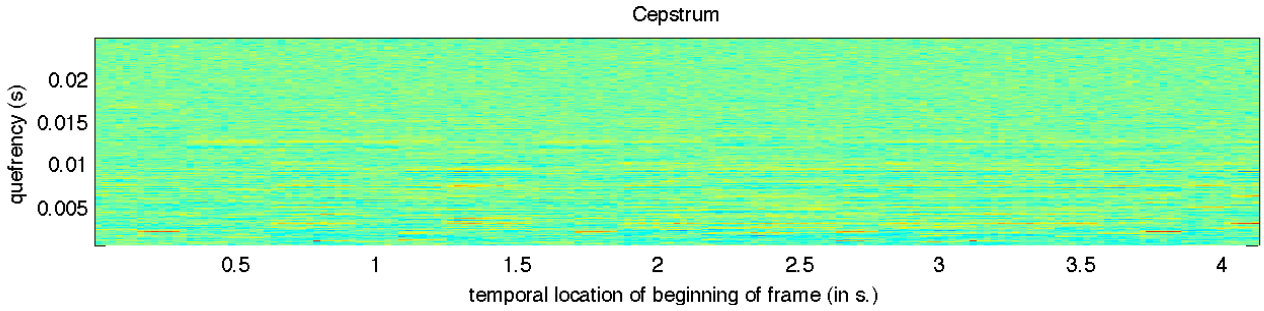
mirflux(s)



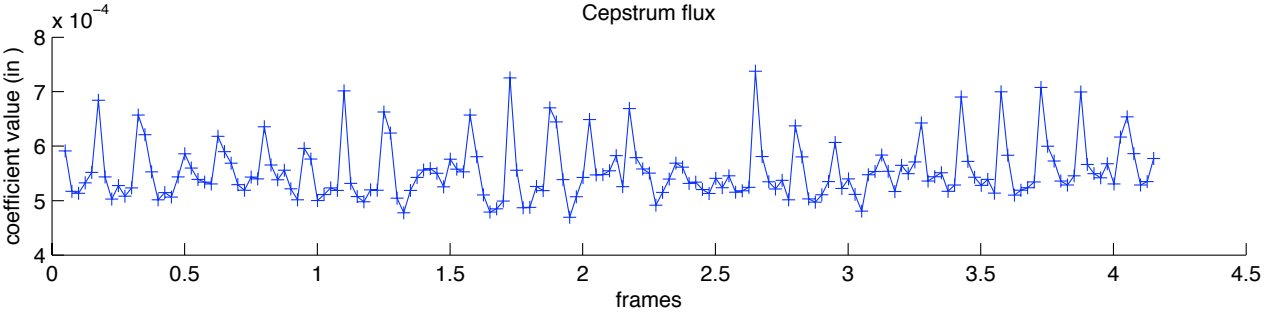
The peaks in the curve indicate the temporal position of important contrast in the spectrogram.

In *MIRtoolbox* fluxes are generalized to any kind of frame-decomposed representation, for instance a cepstral flux:

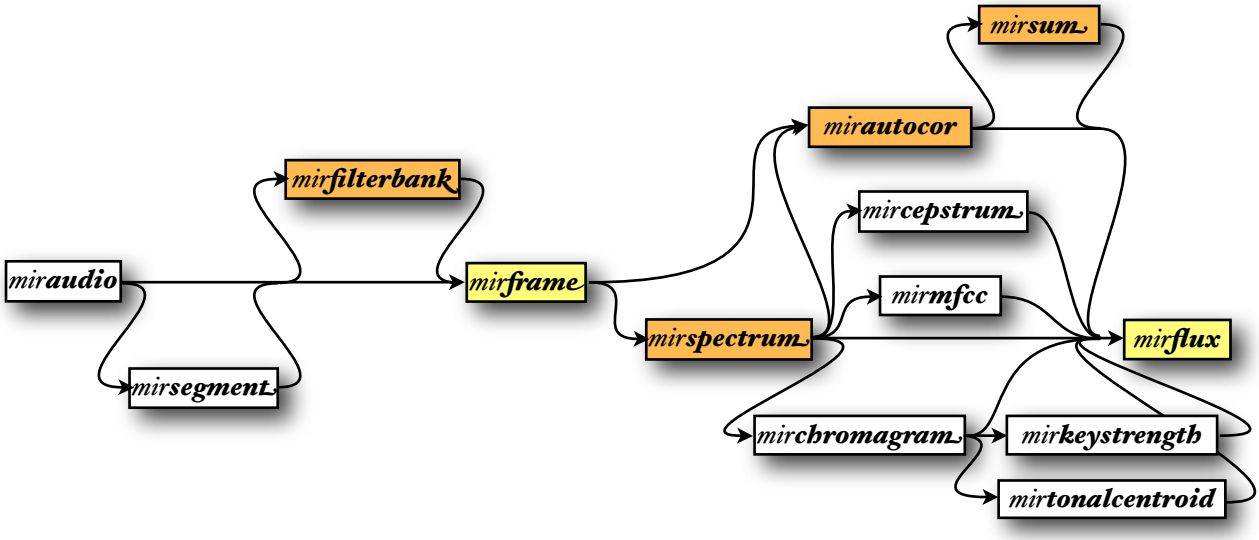
c = mircepstrum(a, 'Frame')



mirflux(c)



FLOWCHART INTERCONNECTIONS



mirflux usually accepts either:

- ***mirspectrum*** frame-decomposed objects.
- ***miraudio*** objects, where the audio waveform can be segmented (using ***mirsegment***), decomposed into channels (using ***mirfilterbank***). The audio waveform is decomposed into frames if it was not decomposed yet, and the default frame parameters – frame length of 50 ms and a hop factor of 0.5 – can be changed using the '***Frame***' option. If the input is a *miraudio* object, the default flux is a spectral flux: i.e., the audio waveform is passed to the *mirspectrum* operator before being fed into *mirflux*.

- **file name** or the ***'Folder'*** keyword: same behavior than for *miraudio* objects;
- ***mirautocor*** frame-decomposed objects;
- ***mircepstrum*** frame-decomposed objects;
- ***mirmfcc*** frame-decomposed objects;
- ***mirchromagram*** frame-decomposed objects;
- ***mirkeystrength*** frame-decomposed objects.

PARAMETERS SPECIFICATION

- *mirflux(x, '**Dist**', d)* specifies the distance between successive frames:
  - *d* = *'**Euclidian**'*: Euclidian distance (Default)
  - *d* = *'**City**'*: City-block distance
  - *d* = *'**Cosine**'*: Cosine distance (or normalized correlation)
- *mirflux(..., '**Inc**')*: Only positive difference between frames are summed, in order to focus on increase of energy solely.
- *mirflux(..., '**Complex**')*, for spectral flux, combines the use of both energy and phase information (Bello et al, 2004).

POST-PROCESSING

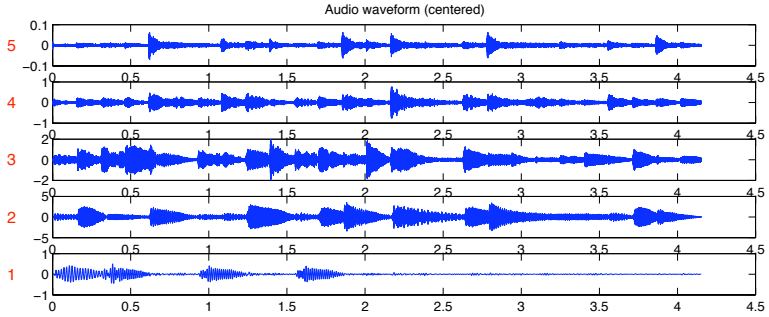
- *mirflux(..., '**Halfwave**')*: performs a half-wave rectification on the result.
- *mirflux(..., '**Median**', l, C)*: removes small spurious peaks by subtracting to the result its median filtering. The median filter computes the point-wise median inside a window of length *l* (in seconds), that includes a same number of previous and next samples. *C* is a scaling factor whose purpose is to artificially rise the curve slightly above the steady state of the signal. If no parameters are given, the default values are: *l* = 0.2 s. and *C* = 1.3
- *mirflux(..., '**Median**', l, C, '**Halfwave**')*: The scaled median filtering is designed to be succeeded by the half-wave rectification process in order to select peaks above the dynamic threshold calculated with the help of the median filter. The resulting signal is called "*detection function*" (Alonso et al., 2003). To ensure accurate detection, the length *l* of the median filter must be longer than the average width of the peaks of the detection function.

mirsum

SUMMATION OF FILTERBANK CHANNELS

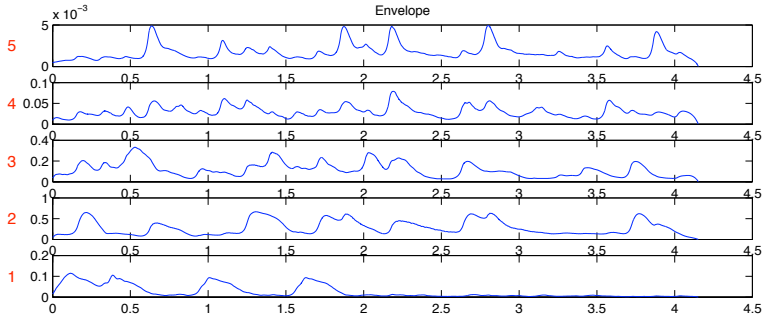
Once an audio waveform is decomposed into channels using a filterbank:

```
f = mirfilterbank(a)
```



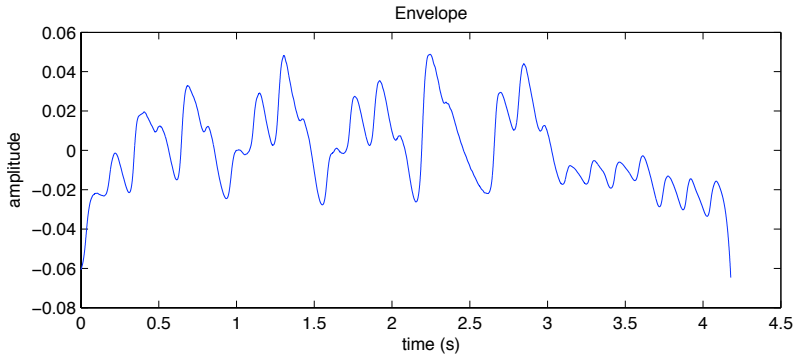
An envelope extraction, for instance, can be computed using this very simple syntax:

```
e = mirenvelope(f)
```



Then the channels can be summed back using the *mirsum* command:

```
s = mirsum(e)
```



The summation can be centered using the command:

s = mirsum(..., 'Center')

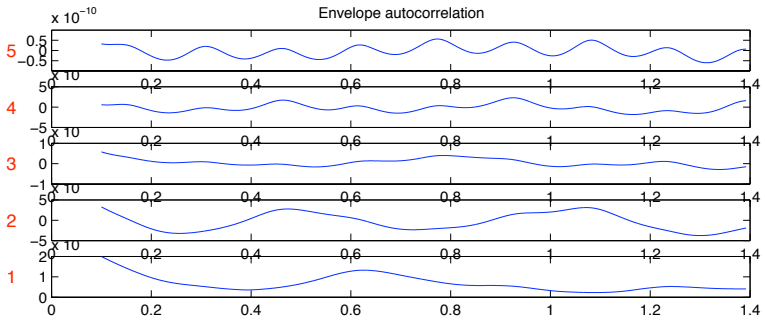
The summation can be divided by the number of channels using the command:

s = mirsum(..., 'Mean')

SUMMARY OF FILTERBANK CHANNELS

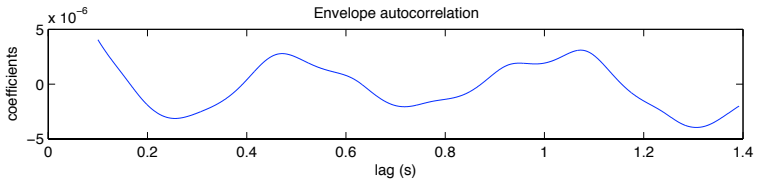
If we compute for instance an autocorrelation from the envelopes:

ac = mirautocor(e)



Then we can sum all the autocorrelation using exactly the same *mirsum* command:

s = mirsum(e)



This summation of non-temporal signals across channels is usually called *summary*.

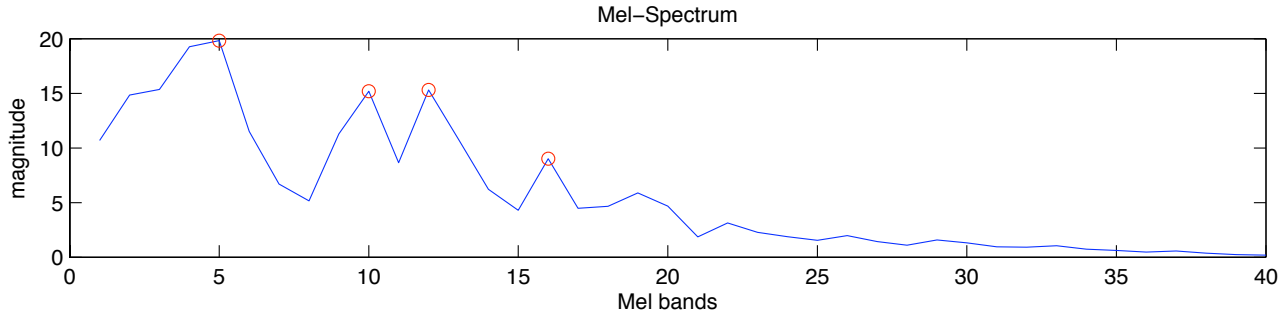
mirpeaks

PEAK PICKING

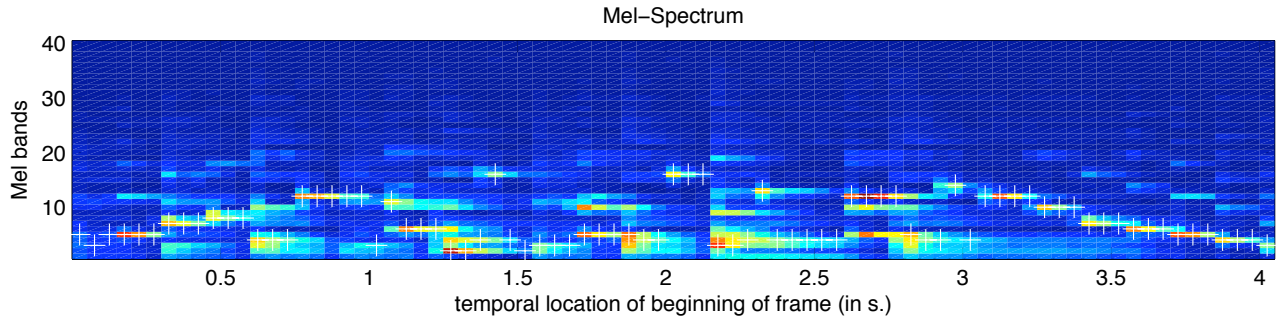
Peaks (or important local maxima) can be detected automatically from any data *x* produced in *MIRtoolbox* using the command

mirpeaks(x)

If *x* is a curve, peaks are represented by red circles:



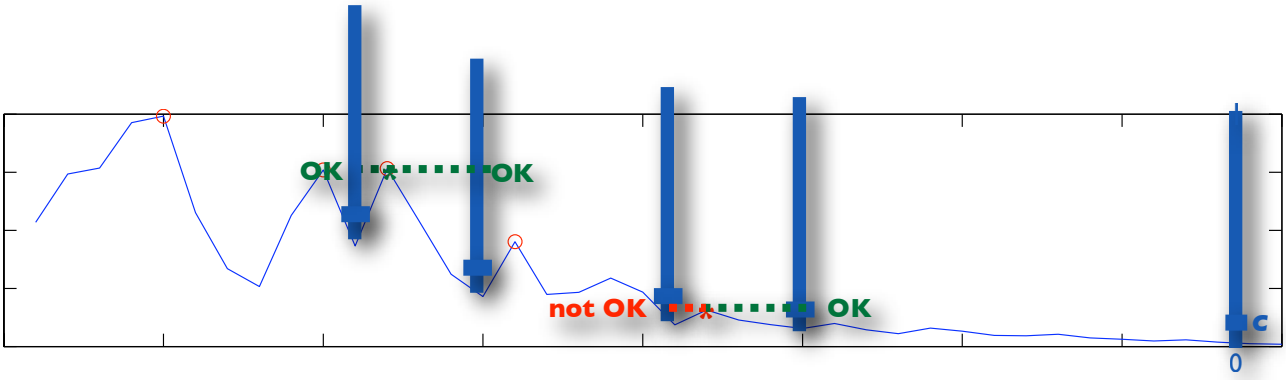
If x is a frame-decomposed matrix, peaks are represented by white crosses:



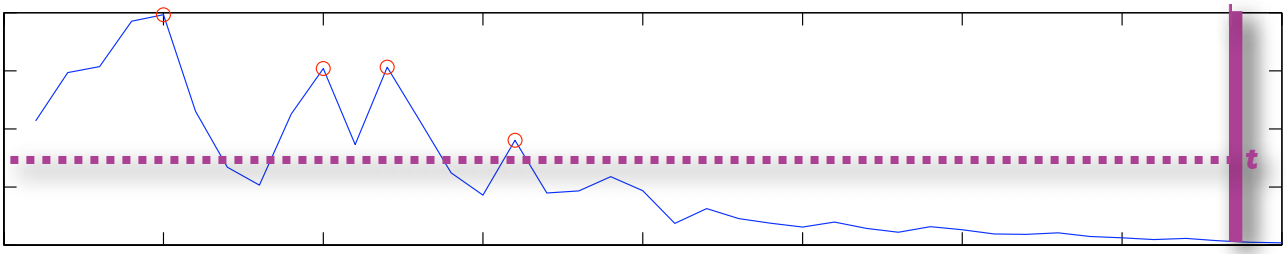
PARAMETERS SPECIFICATION

- *mirpeaks(..., **'Total'**, m)*: only the *m* highest peaks are selected. If *m* = *Inf*, no limitation of number of peaks. Default value: *m* = *Inf*
- Border effects can be specified:
  - *mirpeaks(..., **'NoBegin'**)* does not consider the first sample as a possible peak candidate.
  - *mirpeaks(..., **'NoEnd'**)* does not consider the last sample as a possible peak candidate.
- *mirpeaks(..., **'Order'**, o)* specifies the ordering of the peaks.
  - *o* = *'**Amplitude**'* orders the peaks from highest to lowest (Default choice.)
  - *o* = *'**Abscissa**'* orders the peaks along the abscissa axis.

- *mirpeaks(..., 'Valleys')* detect valleys (local minima) instead of peaks.
- *mirpeaks(..., 'Contrast', cthr)*: A given local maximum will be considered as a peak if the difference of amplitude with respect to both the previous and successive local minima (when they exist) is higher than the threshold *cthr*. This distance is expressed with respect to the total amplitude of the input signal: a distance of 1, for instance, is equivalent to the distance between the maximum and the minimum of the input signal. Default value: *cthr* = 0.1



- *mirpeaks(..., 'SelectFirst', fthr)*: If the *'Contrast'* selection has been chosen, this additional option specifies that when one peak has to be chosen out of two candidates, and if the difference of their amplitude is below the threshold *thr*, then the most ancien one is selected. Option toggled off by default. Default value if toggled on: *fthr* = *cthr*/2
- *mirpeaks(..., 'Threshold', thr)*: A given local maximum will be considered as a peak if its normalized amplitude is higher than this threshold *thr*. A given local minimum will be considered as a valley if its normalized amplitude is lower than this threshold. The normalized amplitude can have value between 0 (the minimum of the signal in each frame) and 1 (the maximum in each frame). Default value: *thr*=0 for peaks, *thr* = 1 for valleys.



- *mirpeaks(..., 'Interpol', i)* estimates more precisely the peak position and amplitude using interpolation. Performed only on data with numerical abscissae axis.
  - *i* = **''**, **'no'**, **'off'**, **0**: no interpolation
  - *i* = *'Quadratic'*: quadratic interpolation. (default value).

- *mirpeaks(..., 'Reso', r)* removes peaks whose abscissa distance to one or several higher peaks is lower than a given threshold. Possible value for the threshold: *r* = '*SemiTone*': ratio between the two peak positions equal to 2^(1/12). By default, out of two conflicting peaks, the higher peak remains. If the keyword '*First*' is added, the peak with lower abscissa value remains instead.

- *mirpeaks(..., 'Pref', c, std)* indicates a region of preference for the peak picking, centered on the abscissa value *c*, with a standard deviation of *std*.

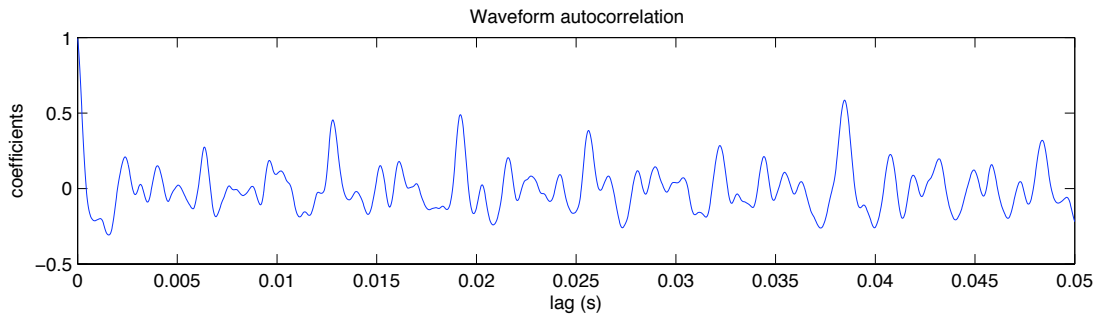
- *mirpeaks(..., 'Nearest', t, s)* takes the peak nearest a given abscissa values *t*. The distance is computed either on a linear scale (*s* = '*Lin*') or logarithmic scale (*s* = '*Log*'). When using the *'Nearest'* option, only one peak is extracted.

The *'Total'* parameter can then be used to indicate the number of peaks to preselect before the '*Nearest*' selection. If '*Total*' was still set to 1, it is then ignored – i.e., forced to *Inf* – in order to preselect all possible peaks.

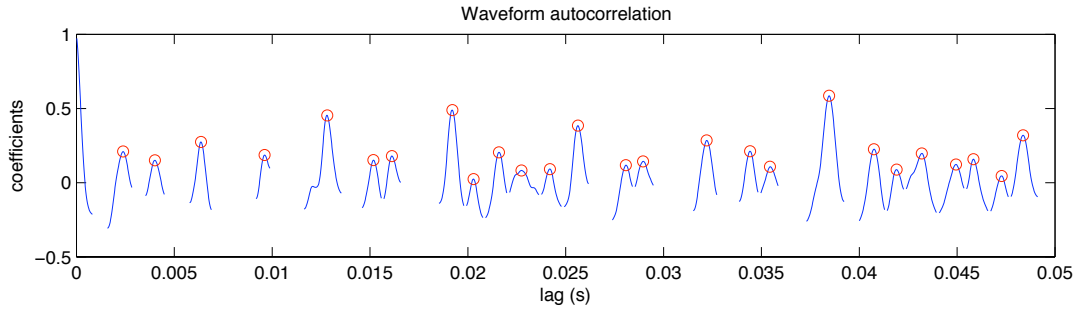
- *mirpeaks(..., 'Normalize', n)* specifies whether frames are normalized globally or individually.
  - *n* = '*Global*' normalizes the whole frames altogether from 0 to 1 (default choice).
  - *n* = '*Local*': normalizes each frame from 0 to 1 separately.

- *mirpeaks(..., 'Extract')* extracts from the curves all the positive continuous segments (or "curve portions") where peaks are located. First, a low-pass filtered version of the curve is computed, on which the temporal span of the positive lobes containing each peak are stored. The output consists of the part of the original non-filtered curve corresponding to the same temporal span. For instance:

ac = mirautocor('ragtime')

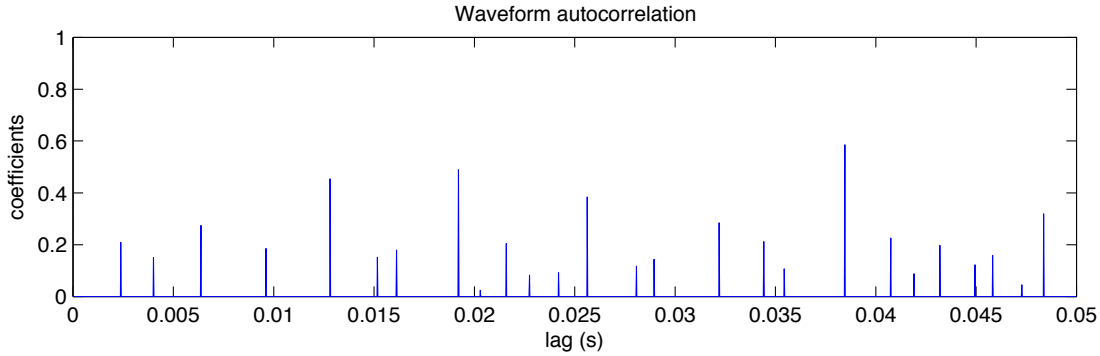


mirpeaks(ac, 'Extract')



- *mirpeaks(..., **'Only'**)*, keeps from the original curve only the data corresponding to the peaks, and zeroes the remaining data.

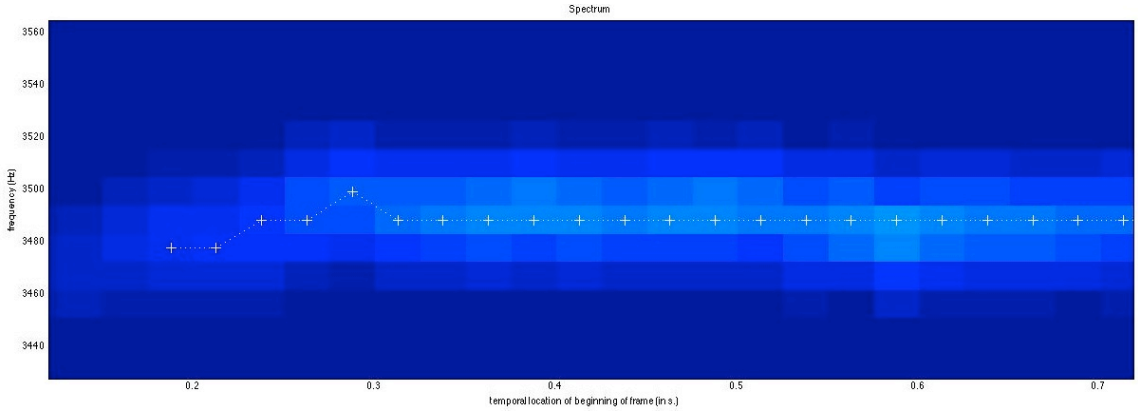
mirpeaks(ac, 'Only')



- *mirpeaks(..., **'Track'**, t)*, where the input is some frame-decomposed vectorial data – such as spectrogram, for instance –, tracks peaks along time using McAulay & Quatieri's (1986) method: lines are drawn between contiguous temporal series of peaks that are sufficiently aligned. If a value *t* is specified, the variation between successive frames is tolerated up to *t*, *expressed using the abscissae unit*. For instance, the figure below shows the result (zoomed) of the following commands:

s = mirspectrum('trumpet', 'Frame');

mirpeaks(s, 'Track', 25)



- *mirpeaks(..., '**CollapseTracks**', t)*, collapses tracks into one single track, and remove small track transitions, of length shorter than *ct* samples. Default value: *ct* = 7.

Accessible Output

cf. §5.2 for an explanation of the use of the *get* method. Specific fields:

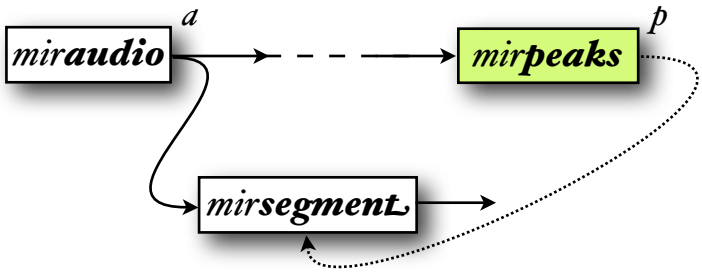
- '***PeakPos***': the abscissae position of the detected peaks, in sample index,
- '***PeakPosUnit***': the abscissae position of the detected peaks, in the default abscissae representation,
- '***PeakPrecisePos***': a more precise estimation of the abscissae position of the detected peaks computed through interpolation, in the default abscissae representation,
- '***PeakVal***': the ordinate values associated to the detected peaks,
- '***PeakPreciseVal***': a more precise estimation of the ordinate values associated to the detected peaks, computed through interpolation,
- '***PeakMode***': the mode values associated to the detected peaks,
- '***TrackPos***': the abscissae position of the peak tracks, in sample index,
- '***TrackPosUnit***': the abscissae position of the peak tracks, in the default abscissae representation,
- '***TrackVal***': the ordinate values of the peak tracks..

mirsegment

SEGMENTATION

- An audio waveform *a* can be segmented using the output *p* of a peak picking from data resulting from *a* itself, using the following syntax:

sg = mirsegment(a, p)



If *p* is a frame-decomposed scalar curve, the audio waveform *a* will be segmented at the middle of each frame containing a peak.

- An audio waveform *a* can also be segmented manually, based on temporal position directly given by the user, in the form:

sg = mirsegment(a, v)

where *v* is an array of numbers corresponding to time positions in seconds.

If *a* is a set of *N* audio waveforms, and if *v* is a matrix of *N* columns, then each column *i* indicates the segmentation points for the waveform *i*. If *v* has only one column, then all waveforms use that same column of segmentation points.

- Automated segmentation methods are provided as well, that can be called using the syntax:

sg = mirsegment(a, m)

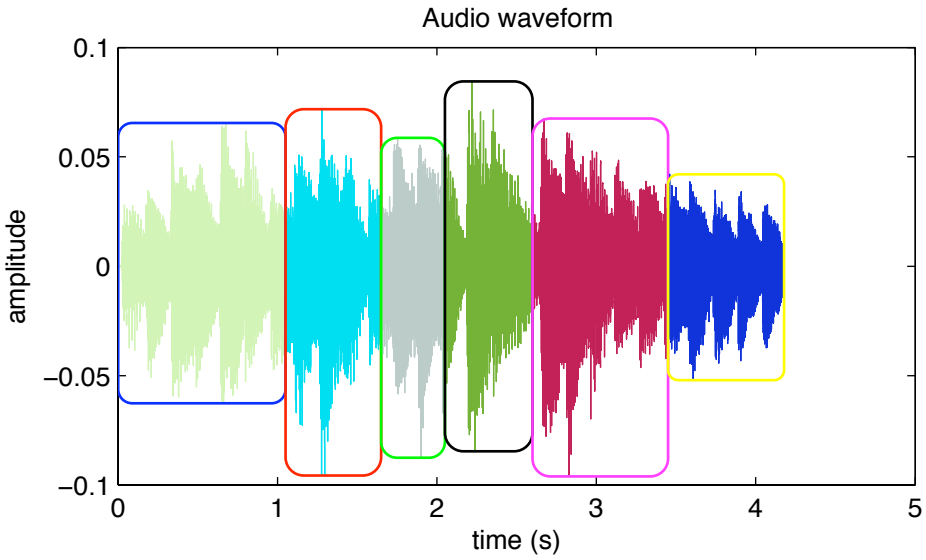
where *m* is the name of one of the following segmentation methods: '*Novelty*' (default, cf. *mirnovelty*), '*HCDF*' (cf. *mirhcdf*) or '*RMS*' (cf. *mirrms*).

mirsegment accepts uniquely as main input a ***miraudio*** objects not frame-decomposed, not channel decomposed, and not already segmented. Alternatively, **file name** or the '***Folder***' keyword can be used as well.

The first argument of the *mirsegment* function is the audio file that needs to be segmented. It is possible for instance to compute the segmentation curve using a downsampled version of the signal and to perform the actual segmentation using the original audio file.

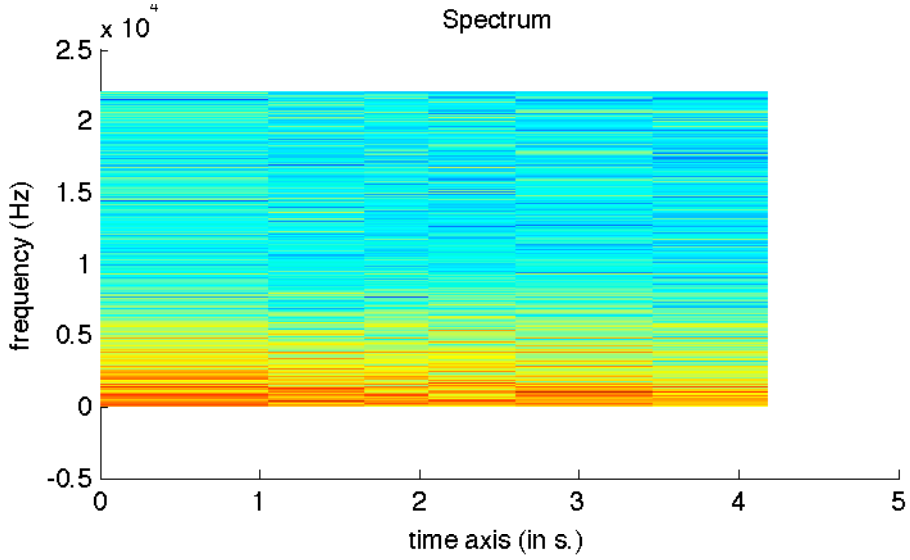
EXAMPLE

sg = mirsegment('ragtime', 'Novelty', 'KernelSize', 32)



The output can be sent to any further analysis, for instance:

sp = mirspectrum(sg, 'dB')



cf. §5.2 for an explanation of the use of the *get* method.

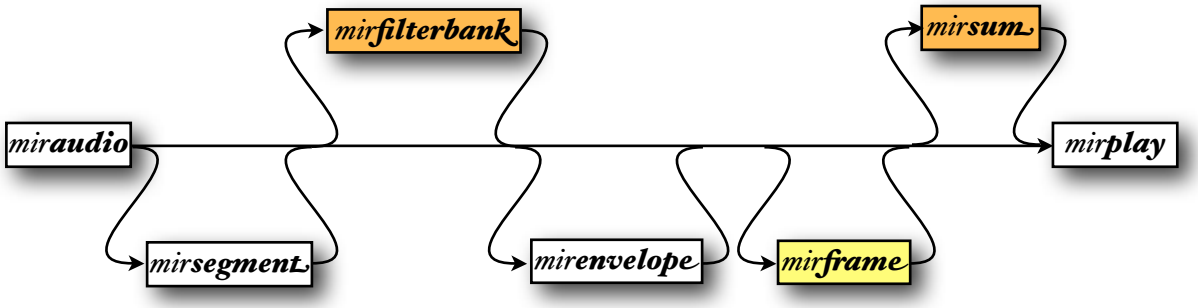
- '***FramePos***': For segmented data, this returns a cell array where each cell is a matrix containing the starting and ending temporal positions of the frames in each successive segment. When there is no frame decomposition, each cell contains simply the starting and ending time of each successive segment.

*mir**play***

Sonification of the result

Certain classes of temporal data can be sonified:

- ***miraudio*** objects: the waveform is directly played, and
  - if the audio waveform is segmented (using ***mirsegment***), segments are played successively with a short burst of noise in-between;
  - if the audio waveform is decomposed into channels (using ***mirfilterbank***), channels are played successively from low to high register;
  - if the audio is decomposed into frames (using ***mirframe*** or the '***Frame***' option, with by default a frame length of 50 ms and half overlapping), frames are played successively;
- **file name** or the '***Folder***' keyword: same behavior than for *miraudio* objects;
- ***mirenvelope*** objects (frame-decomposed or not) are sonified using a white noise modulated in amplitude by the envelope, and
  - if peaks have been picked on the envelope curve (using ***mirpeaks***), they are sonified using a short impulse sound;
- ***mirpitch*** results: each extracted frequency is sonified using a sinusoid.



Options

- *mirplay(..., '**Channel**', i)* plays the channel(s) of rank(s) indicated by the array *i*.
- *mirplay(..., '**Segment**', k)* plays the segment(s) of rank(s) indicated by the array *k*.
- *mirplay(..., '**Sequence**', l)* plays a sequence of audio files using the order indicated by the array *l*.
- *mirplay(..., '**Increasing**', d)* plays the sequences in increasing order of *d*, which can be either an array of number or a *mirscalar* data (i.e., a scalar data returned by *MIRtoolbox*).

- *mirplay(..., '**Decreasing**', d)* plays the sequences in decreasing order of *d*, which can be either an array of number or a *mirscalar* data (i.e., a scalar data returned by *MIRtoolbox*).
- *mirplay(..., '**Every**', s)* plays every *s* sequence, where *s* is a number indicating the step between sequences.
- *mirplay(...,'**Burst**', 'No')* toggles off the burst sound between segments.

Example:

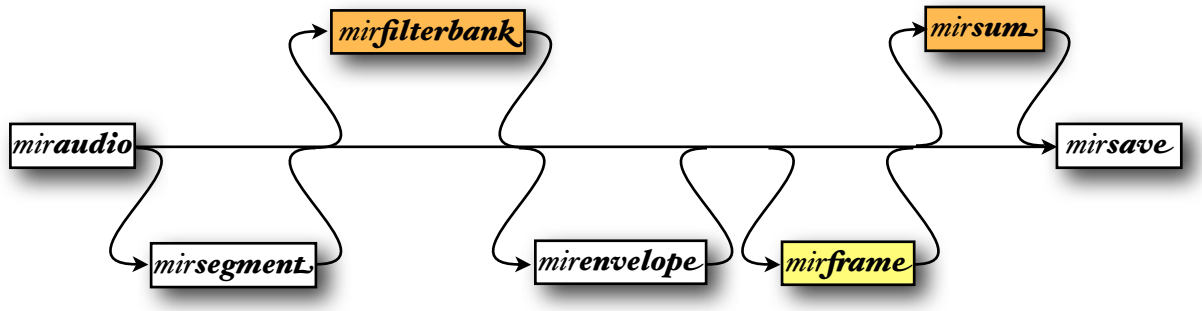
```
e = mirenvelope('Folder');

rms = mirrms('Folder');

mirplay(e, 'increasing', rms, 'every',5)
```

mirsave

Saving audio rendering into files

Certain classes of temporal data can be saved:

- ***miraudio*** objects: the waveform is directly saved, and
  - if the audio waveform is segmented (using ***mirsegment***), segments are concatenated with a short burst of noise in-between;
  - if the audio waveform is decomposed into channels (using ***mirfilterbank***), each channel is saved in a separate file;
  - if the audio is decomposed into frames (using ***mirframe*** or the '***Frame***' option, with by default a frame length of 50 ms and half overlapping), frames are concatenated;
- **file name** or the '***Folder***' keyword: same behavior than for *miraudio* objects;
- ***mirenvelope*** objects (frame-decomposed or not) are sonified using a white noise modulated in amplitude by the envelope,
- ***mirpitch*** results: each extracted frequency is sonified using a sinusoid.



Options

- The name and extension of the saved file can be specified in different ways, as shown in the tables below.
  - By default, the files are saved in WAV format, using the extension '*.mir.sav*' in order to lower the risk of overwriting original audio files.
  - If the string '***.au***' is indicated as second argument of *mirsave*, the audio will be saved in AU format.
  - A string can be indicated as second argument of *mirsave*.

- If the *miraudio* object to be saved contains only one audio file, the specified string will be used as the name of the new audio file.
- If the *miraudio* object to be saved contains several audio files, the specified string will be concatenated to the original name of each audio file.

- If the second argument of *mirsave* is a string ended by '*.au*', the file name will follow the convention explained in the previous point, and the files will be saved in AU format.

a = miraudio('mysong.au')	*mysong.au*
mirsave(a)	*mysong**.mir.wav***
*mirsave(a,'**new**')*	***new.wav***
*mirsave(a,'**.au**')*	*mysong**.mir.au***
*mirsave(a,'**new.au**')*	***new.au***

Diverse ways of saving into an audio file

a = miraudio('Folder')	*song1.wav*	*song2.wav*	*song3.au*
mirsave(a)	*song1**.mir**.wav*	*song2**.mir**.wav*	*song3**.mir.wav***
*mirsave(a,'**new**')*	*song1**new**.wav*	*song2**new**.wav*	*song3**new.wav***
*mirsave(a,'**.au**')*	*song1**.mir.au***	*song2**.mir.au***	*song3**.mir.au***
*mirsave(a,'**new.au**')*	*song1**new.au***	*song2**new.au***	*song3**new.au***

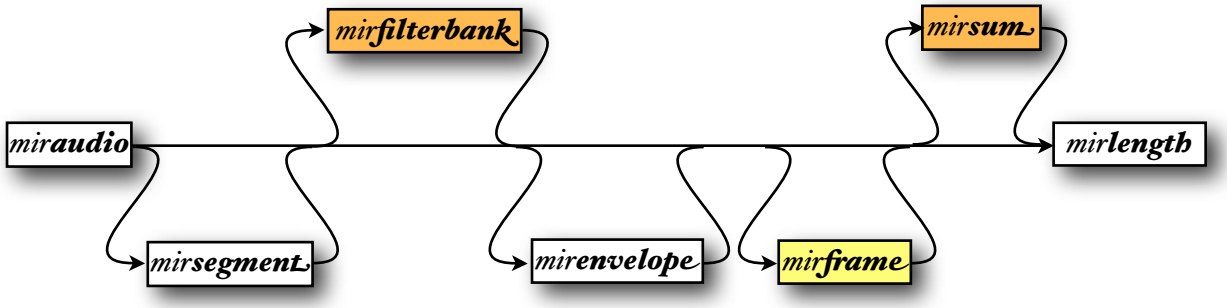
Diverse ways of saving as a batch of audio files

- *mirsave(a, filename, '**SeparateChannels**')* save each separate channel in a different file. The channel number is added to the file name, before any file extension.

mirlength

TEMPORAL LENGTH OF SEQUENCES

mirlength returns the temporal length of the temporal sequence given in input, which can be either an audio waveform (*miraudio*) or an envelope curve (*mirenvelope*). If the input was decomposed into segments (*mirsegment*), *mirlength* returns a curve indicating the series of temporal duration associated with each successive segment.



OPTIONS

- *mirlength(..., 'Unit', u)* specifies the length unit. The possible values are:
  - *u* = *'Second'*: duration in seconds (Default choice).
  - *u* = *'Sample'*: length in number of samples.

3. FEATURE EXTRACTORS

The musical feature extractors can be organized along main musical dimensions: dynamics, rhythm, timbre, pitch and tonality.

3.1. Dynamics

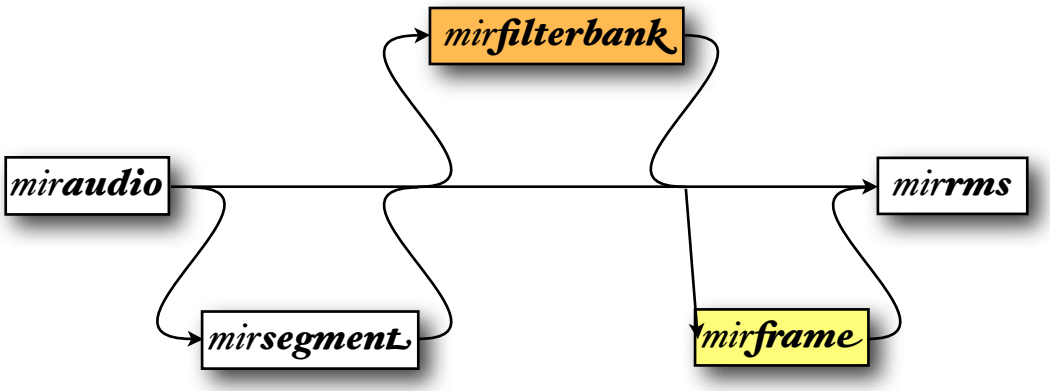
mirrms

ROOT-MEAN-SQUARE ENERGY

The global energy of the signal x can be computed simply by taking the root average of the square of the amplitude, also called root-mean-square (RMS):

$$x_{\mathrm{rms}} = \sqrt{\frac{1}{n}\sum_{i=1}^{n} x_i^2} = \sqrt{\frac{x_1^2 + x_2^2 + \cdots + x_n^2}{n}}$$

FLOWCHART INTERCONNECTIONS



mirrms accepts as input data type either:

- ***miraudio*** objects, where the audio waveform can be segmented (using ***mirsegment***), decomposed into channels (using ***mirfilterbank***), and/or decomposed into frames (using ***mirframe*** or the '***Frame***' option, with by default a frame length of 50 ms and half overlapping),
- **file name** or the '***Folder***' keyword.

The following command orders the computation of the RMS related to a given audio file:

mirrms('ragtime')

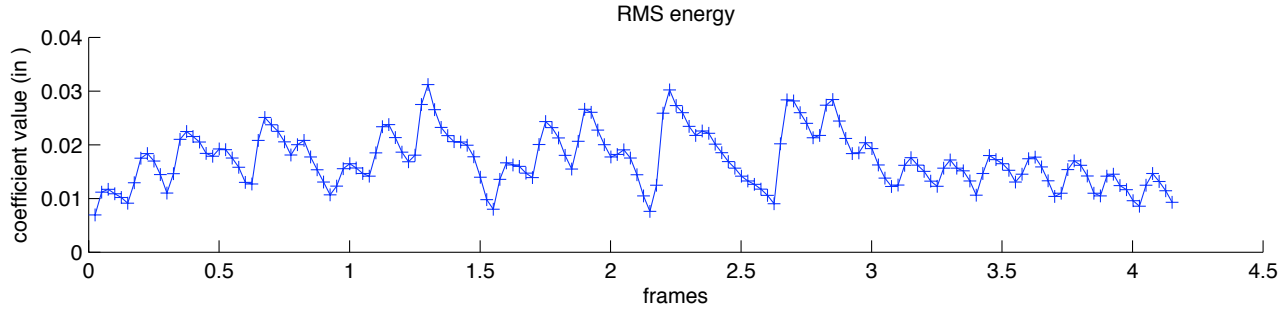
which produce the resulting message in the *Command Window*:

```
The RMS energy related to file ragtime is 0.017932
```

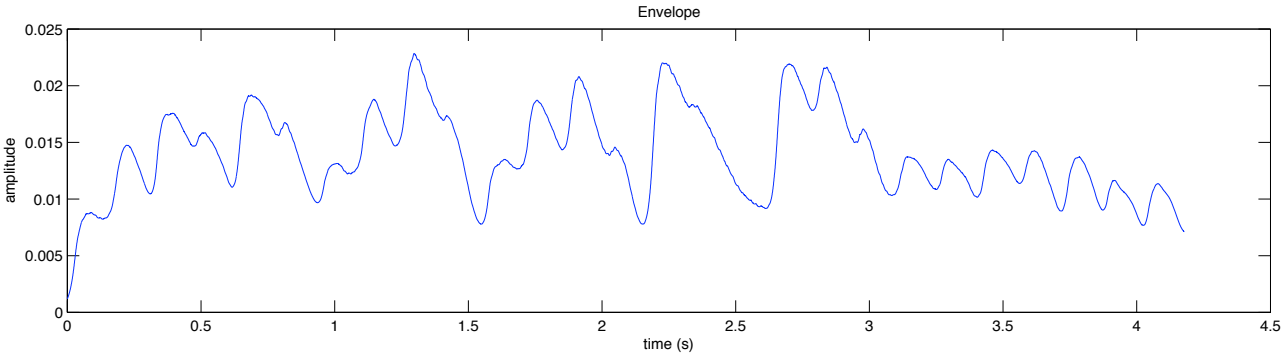
If we know ask for a frame-decomposed computation of RMS:

mirrms('ragtime', 'Frame')

we obtain a temporal evolution of the energy:



We can note that this energy curve is very close to the envelope:



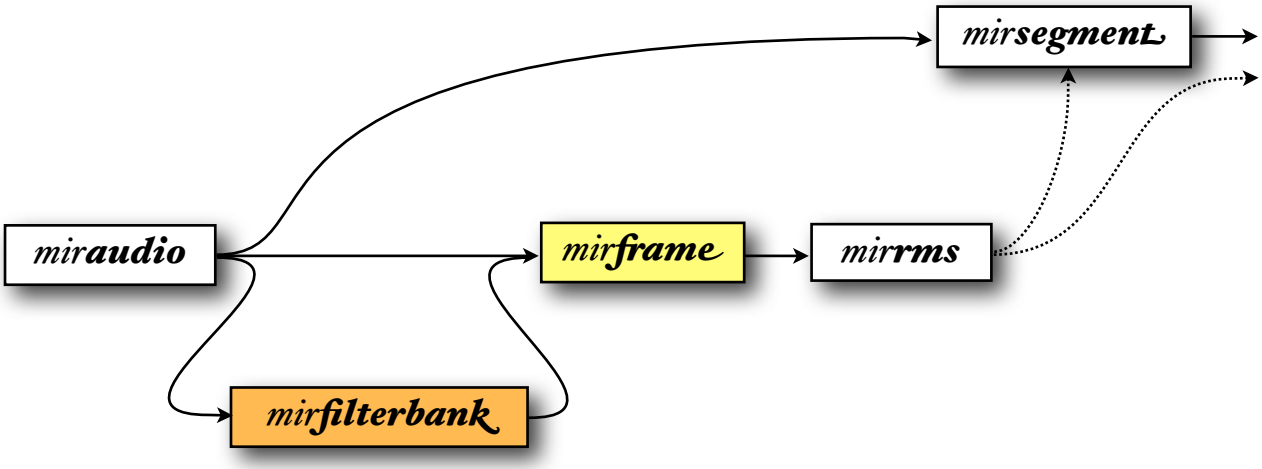
mirsegment(..., 'RMS')

Segmentation at positions of long silences. A frame decomposed RMS is computed using *mirrms* (with default options), and segments are selected from temporal positions where the RMS rises to a given '*On*' threshold, until temporal positions where the RMS drops back to a given '*Off*' threshold.

Options

- *mirsegment(...,'**Off**', t1)* specifies the RMS '*Off*' threshold. Default value: *t1* = .01
- *mirsegment(...,'**On**', t2)* specifies the RMS '*On*' threshold. Default value: *t2* = .02

Flowchart Interconnections



mirsegment accepts uniquely as main input a ***miraudio*** objects not frame-decomposed, not channel decomposed, and not already segmented. Alternatively, **file name** or the '***Folder***' keyword can be used as well.

mirsegment(..., 'RMS') can return several outputs:

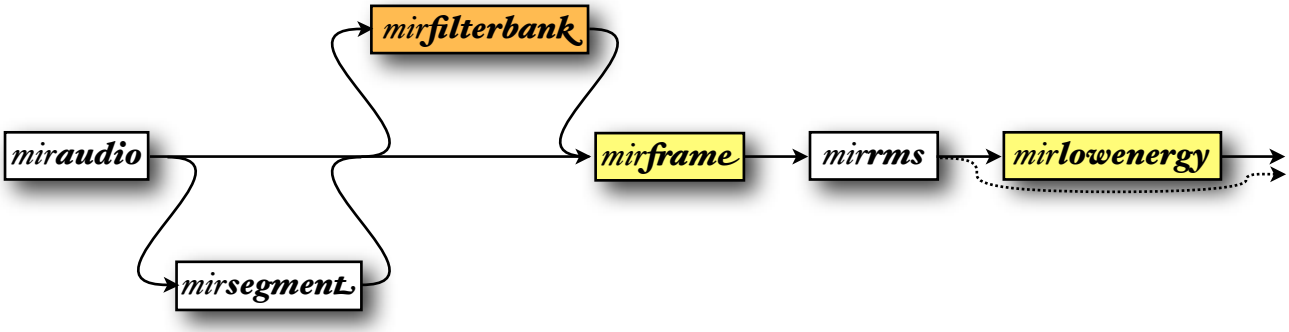
1. the segmented audio waveform itself,
2. the RMS curve (***mirrms***).

mirlowenergy

Description

The energy curve can be used to get an assessment of the temporal distribution of energy, in order to see if its remains constant throughout the signal, or if some frames are more contrastive than others. One way to estimate this consists in computing the low energy rate, i.e. the percentage of frames showing less-than-average energy (Tzanetakis and Cook, 2002).

Flowchart Interconnections



mirlowenergy accepts as input data type either:

- ***mirrms* frame-decomposed** data,
- ***miraudio*** objects, where the audio waveform can be segmented (using ***mirsegment***), decomposed into channels (using ***mirfilterbank***). The audio waveform is decomposed into frames if it was not decomposed yet, and the default frame parameters – frame length of 50 ms and half overlapping – can be changed using the '***Frame***' option.
- **file name** or the '***Folder***' keyword: same behavior than for *miraudio* objects.

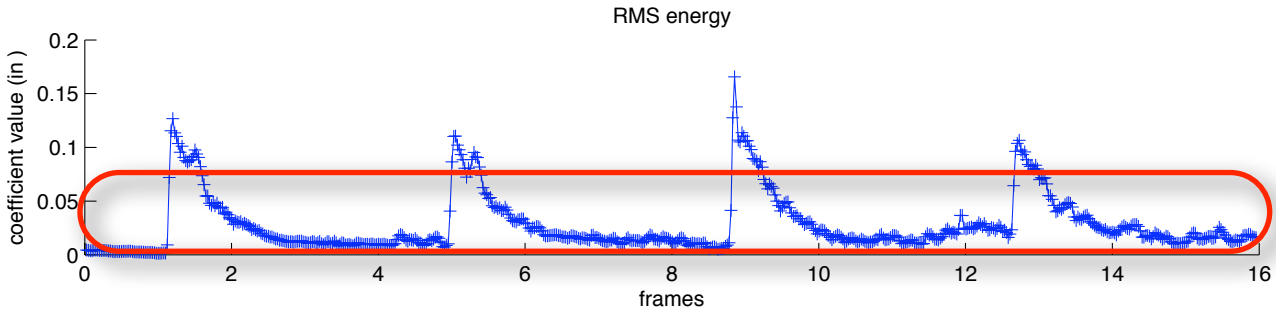
mirlowenergy can return several outputs:

1. the low-energy rate itself and
2. the ***mirrms* frame-decomposed** data.

Examples

If we take for instance this energy curve:

r1 = mirrms('a1', 'Frame')



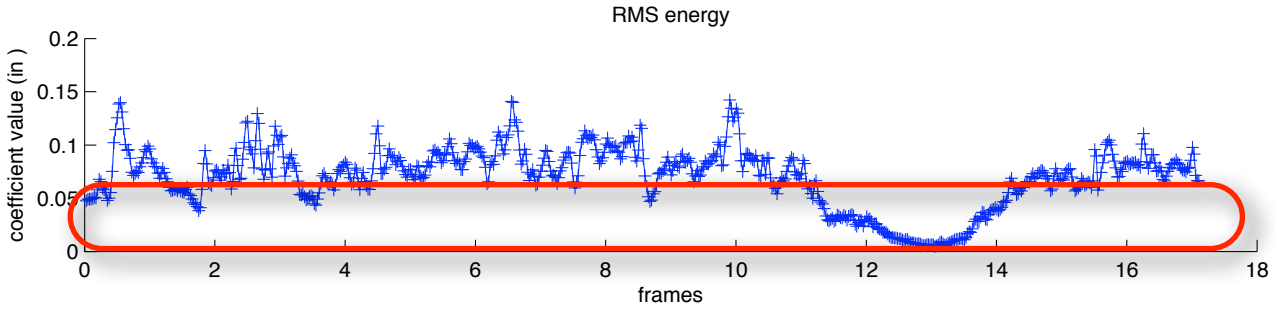
We can see that due to some rare frames containing particularly high energy, most of the frames are below the average RMS. And indeed if we compute the low-energy rate

mirlowenergy(r1)

we obtain the value 0.71317.

For this opposite example:

r2 = mirrms('a2', 'Frame')



there are two kind of frames basically, those that have quite constant high energy, and fewer that have very low energy. Hence most of the frames are over the average energy, leading to a low low-energy rate:

mirlowenergy(r2)

equal to 0.42398.

OPTIONS

- *mirlowenergy(..., '**Threshold**', t)* expressed as a ratio to the average energy over the frames. Default value: t = 1
- *mirlowenergy(..., '**ASR**')* computes the Average Silence Ratio (Feng, Zhuang, Pan, 2003), which corresponds to a RMS without the square-root, and a default threshold set to $t = .5$

3.2. Rhythm

The estimation of rhythmicity in the audio signal can be performed using the basic operators we introduced previously.

mirfluctuation

RHYTHMIC PERIODICITY ALONG AUDITORY CHANNELS

One way of estimating the rhythmic is based on spectrogram computation transformed by auditory modeling and then a spectrum estimation in each band (Pampalk et al., 2002). The implementation proposed in *MIRtoolbox* includes a subset of the series of operations proposed in Pampalk et al.:

- First the spectrogram is computed on frames of 23 ms and half overlapping, then the Terhardt outer ear modeling is computed, with Bark-band redistribution of the energy, and estimation of the masking effects, and finally the amplitudes are computed in dB scale:

s = mirspectrum(..., 'Frame', .023, .5, 'Terhardt', 'Bark', 'Mask', 'dB')

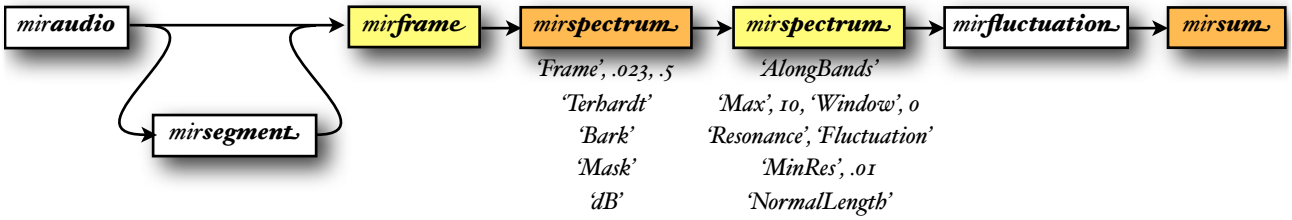
- Then a FFT is computed on each Bark band, from 0 to 10 Hz, with a resolution specified by the '*MinRes*' option (default: .01 Hz). The amplitude modulation coefficients are weighted based on the psychoacoustic model of the fluctuation strength (Fastl, 1982). We can see in the matrix the rhythmic periodicities for each different Bark band.

f = mirspectrum(s, 'AlongBands', 'Max', 10, 'Window', 0, 'Resonance', 'Fluctuation', 'NormalLength')

- *mirfluctuation(..., '**Summary**')* subsequently sums the resulting spectrum across bands, leading to a spectrum summary, showing the global repartition of rhythmic periodicities:

mirsum(f)

Flowchart Interconnections



mirfluctuation accepts as input data type either:

- ***mirspectrum* frame-decomposed** objects (i.e., spectrograms),
- ***miraudio*** objects, where the audio waveform can be segmented (using ***mirsegment***). The audio waveform is decomposed into frames if it was not decomposed yet, using the following frame parameters: frame length of 23 ms and half overlapping.
- **file name** or the ***'Folder'*** keyword: same behavior than for *miraudio* objects.

If you need a frame-decomposed fluctuation curve, showing the temporal evolution of fluctuation frame after frame, you can *not* use *mirframe* or the *'Frame'* option. Why? because *mirfluctuation* already implies frame decomposition from the start, so *mirframe* will not toggle on the frame decomposition (already toggled on), but just control the frame parameters.

If you want to get the temporal evolution of fluctuation, first perform a manual segmentation (for instance, every 10 s). You can put any large number as last argument (100s, or whatever).

```
s = mirsegment('test', 0:10:100);
```

Then you can compute the fluctuation of each successive segment separately:

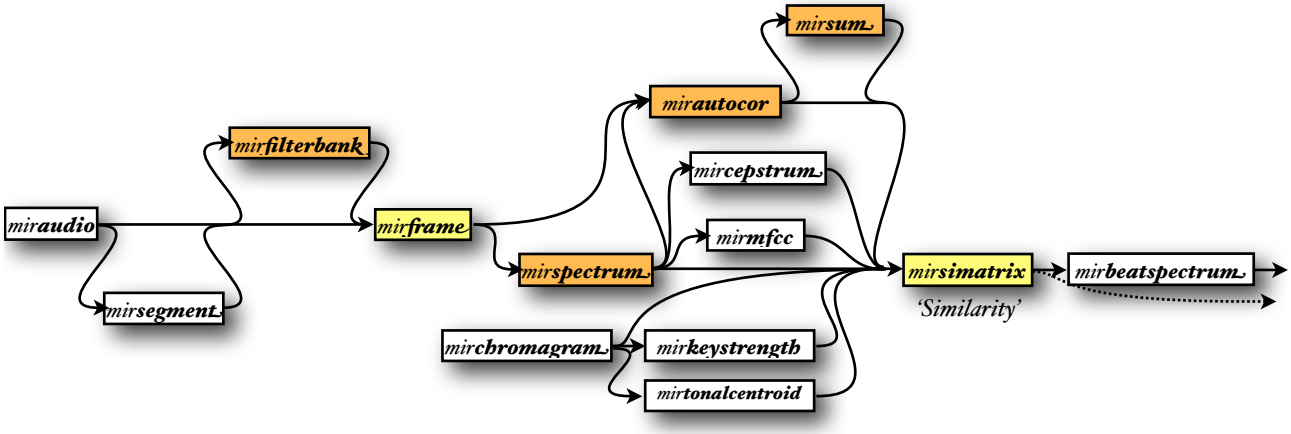
```
fluct = mirfluctuation(s);
```

mirbeatspectrum

Beat spectrum

The beat spectrum has been proposed as a measure of acoustic self-similarity as a function of time lag, and is computed from the similarity matrix (cf. *mirsimatrix*) (Foote, Cooper and Nam, 2002).

Flowchart Interconnections



One parameter related to ***mirsimatrix*** is accessible in *mirbeatspectrum*:

- '***Distance***'.

mirbeatspectrum accepts either:

- ***mirsimatrix*** objects,
- ***mirspectrum*** frame-decomposed objects,
- ***miraudio*** objects: in this case, the similarity matrix will be based on the mfcc (***mirmfcc***), computed from ranks 8 to 33. The audio waveform is decomposed into frames if it was not decomposed yet, and the default frame parameters – frame length of 25 ms with 10 ms overlapping – can be changed using the '***Frame***' option. **file name** or the '***Folder***' keyword: same behavior as for *miraudio* objects,
- other frame-decomposed analysis.

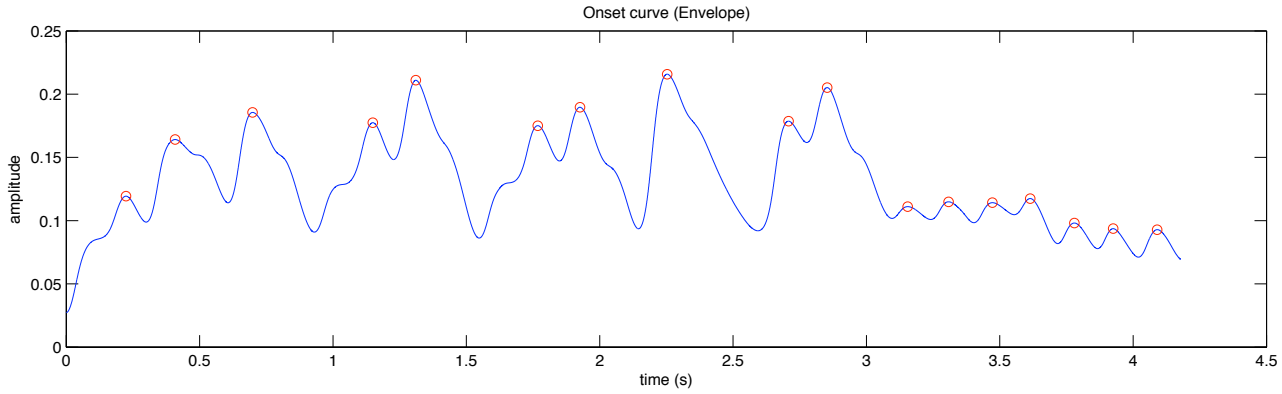
mirbeatspectrum can return several outputs:

1. the beat spectrum curve itself, and
2. the similarity matrix (***mirsimatrix***).

mironsets

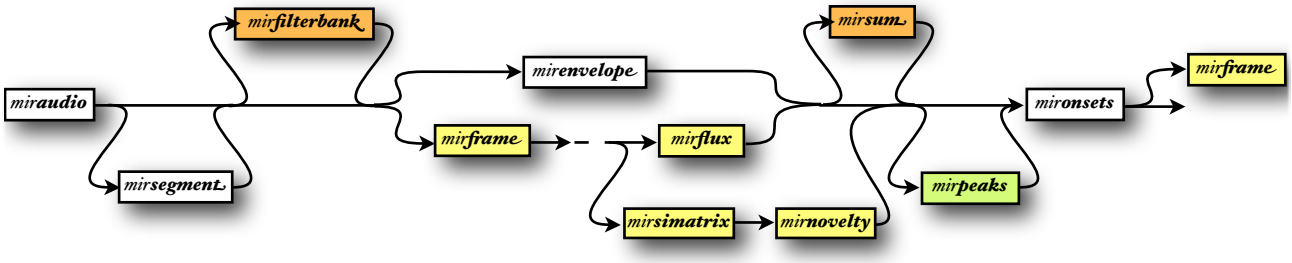
Estimation of notes onset time

Another way of determining the tempo is based on first the computation of an onset detection curve, showing the successive bursts of energy corresponding to the successive pulses. A peak picking is automatically performed on the onset detection curve, in order to show the estimated positions of the notes.



mironsets('ragtime')

Flowchart Interconnections



The onset detection curve can be computed in various ways:

- *mironsets(..., '**Envelope**')* computes an amplitude envelope, using ***mirenvelope*** (default choice). The envelope extraction can be specified, as in *mirenvelope*:
  - either the '***Spectro***' option (default):
    - *mironsets(...,'**SpectroFrame**', fl, fh)* species the frame length *fl* (in s.) and the hop factor *fh* (as a value between 0 and 1). Default values: *fl* = .1 s., *fh* = .1
    - the frequency reassigment method can be specified: '***Freq***' (default), '***Mel***', '***Bark***' or '***Cents***' (cf. *mirspectrum*).

- or the '***Filter***' option: Related options in *mirenvelope* can be specified: '***FilterType***', '***Tau***', '***PreDecim***' with same default value than for *mirenvelope*.
  - *mironsets(...,'**Filterbank**', N)* specifies the number of channels for the filterbank decomposition (***mirfilterbank***): the default value being N = 40. N = 0 toggles off the filterbank decomposition.
  - *mironsets(...,'**FilterbankType**', t)* specifies the type of filterbank decomposition (cf. *mirfilterbank*).
- *mironsets(..., '**Sum**', 'off')* toggles off the channel summation (***mirsum***) that is performed by default.
- Other available options, related to *mirenvelope*: '***HalfwaveCenter***', '***Log***', '***Mu***', '***Power***', '***Diff***', '***HalfwaveDiff***', '***Lambda***', '***Center***', '***Smooth***', '***PostDecim***', '***Sampling***', '***UpSample***', all with same default as in *mirenvelope.*

- *mironsets(..., '**SpectralFlux**')* computes a spectral flux. Options related to ***mirflux*** can be passed here as well:
  - '***Inc***' (toggled on by default here),
  - '***Halfwave***' (toggled on by default here),
  - '***Complex***' (toggled off by default as usual),
  - '***Median***' (toggled on by default here, with same default parameters than in *mirflux*).
- *mironsets(..., '**Pitch**')* computes a frame-decomposed autocorrelation function (***mirautocor***), of same default characteristics than those returned by ***mirpitch*** – with however a range of frequencies set by the following options:
  - '***Min***' (set by default to 30 Hz),
  - '***Max***' (set by default to 1000 Hz),

and subsequently computes the novelty curve of the resulting similatrix matrix. Option related to ***mirnovelty*** can be passed here as well:

- '***KernelSize***' (set to 32 samples by default).

mironsets accepts as input data type either:

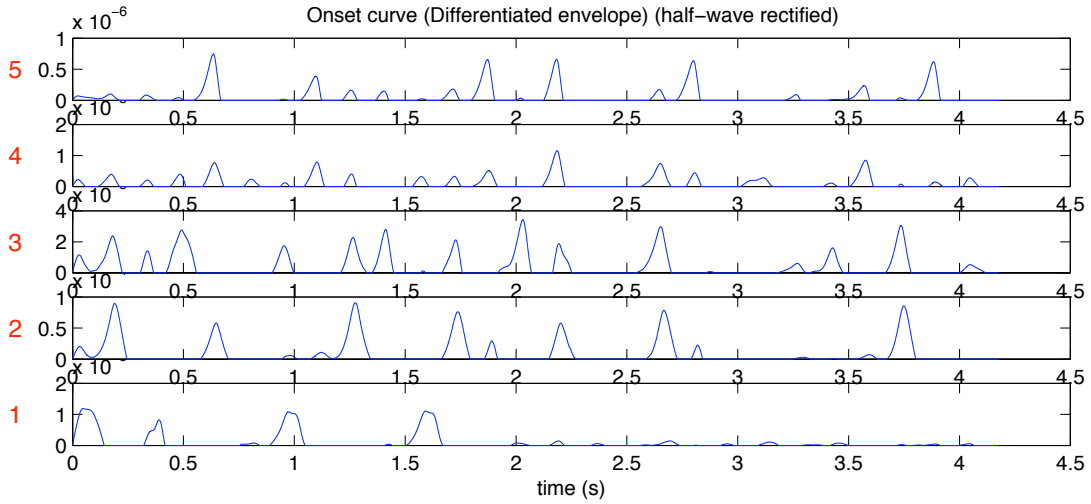
- envelope curves (resulting from ***mirenvelope***),
- any scalar object, in particular:

  - fluxes (resulting from ***mirflux***)
  - novelty curves (resulting from ***mirnovelty***)
- similatrix matrices (resulting from ***mirsimatrix***): its novelty is automatically computed, with a '*KernelSize*' of 32 samples.
- ***miraudio*** objects, where the audio waveform can be:
  - segmented (using ***mirsegment***),
  - decomposed into channels (using ***mirfilterbank***),
  - decomposed into frames or not (using ***mirframe***):
    - if the audio waveform is decomposed into frames, the onset curve will be based on the spectral flux;
    - if the audio waveform is not decomposed into frames, the default onset curve will be based on the envelope;
- **file name** or the '***Folder***' keyword: same behavior than for *miraudio* objects,
- any other object: it is decomposed into frames (if not already decomposed) using the parameters specified by the '*Frame*' option; the flux will be automatically computed by default, or the novelty (if the 'Pitch' option has been chosen).

Example

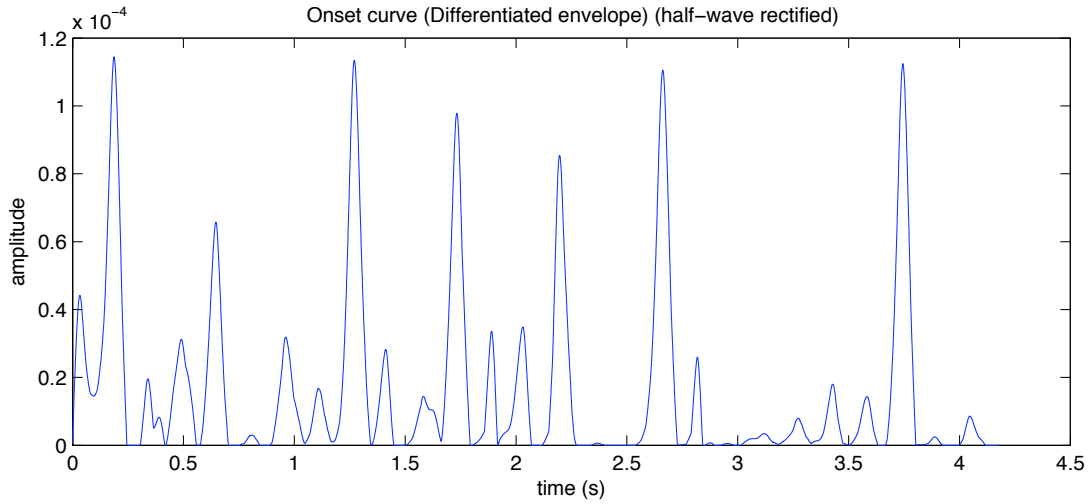
Differentiating the envelope using the '*Diff*' option highlights the difference of energy. By subsequently applying a halfwave rectification of the result ('*HalfwaveDiff*'), bursts of energy are emphasized:

For the previous example (cf. figure above) we obtain now for the differentiated envelopes the following representation:



o = mironsets('ragtime', 'Diff', 'Sum', 'no', 'Filterbank', 5, 'Halfwavediff', 'Detect', 'no')

And once the enveloped are summed:



mirsum(o)

ONSET DETECTION

- *mironsets(..., 'Detect', d)* specifies options related to the peak picking from the onset detection curve:
  - *d* = '*Peaks*' (default choice): local maxima are chosen as onset positions;
  - *d* = '*Valleys*': local minima are chosen as onset positions;
  - *d* = 0, or '*no*', or '*off*': no peak picking is performed.

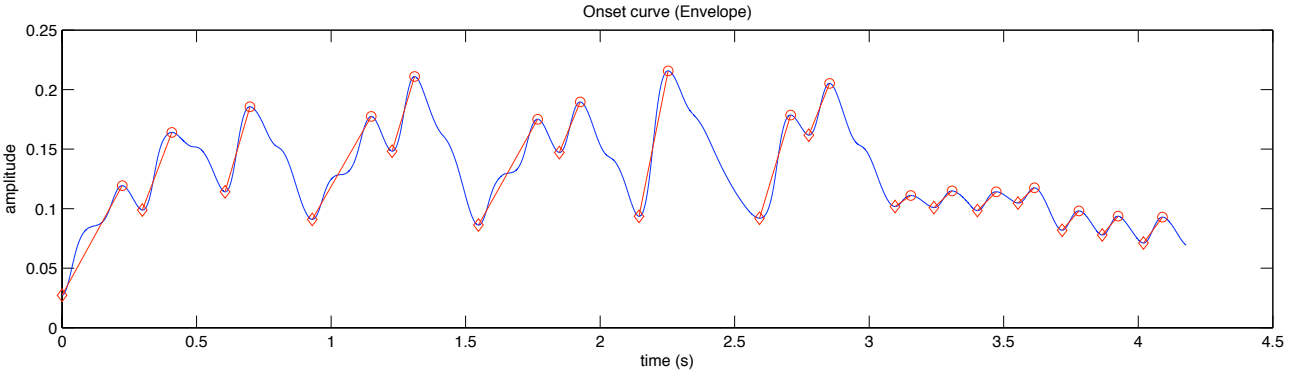
Options associated to the ***mirpeaks*** function can be specified as well. In particular:

- *mironsets(..., 'Contrast', c)* with default value here *c* = .01,
- *mironsets(..., 'Threshold', t)* with default value here *t* = 0.

ATTACK AND RELEASE

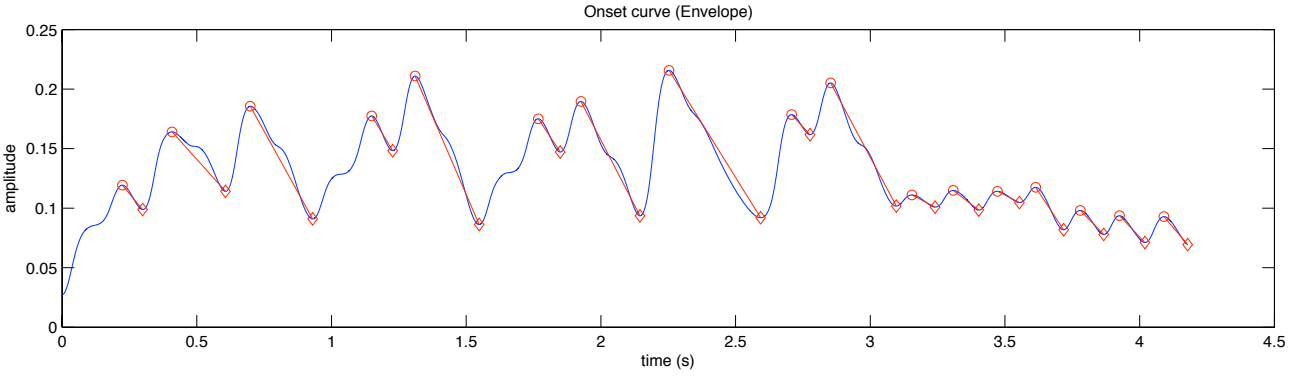
The maxima of the onset detection curve show the positions of the note onsets, but more precisely the end of the attack phase. The '*Attack*' and '*Release*' options estimate the beginning of the attack phase and the end of the release phase of each note by searching for the local minimum before and after each peak.

- *mironsets(..., 'Attack')* (or '*Attacks*') detects attack phases.



mironsets('ragtime', 'attacks')

- *mironsets*(..., '*Release*', *r*) (or '*Releases*') detects release phases.

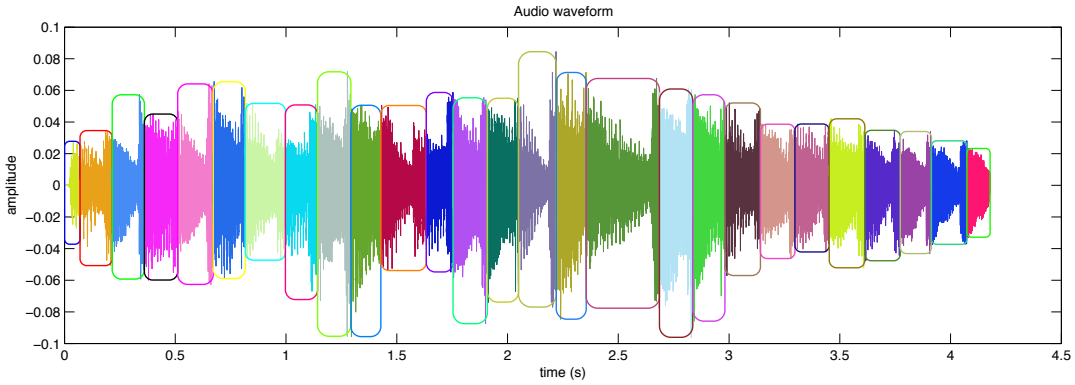


mironsets('ragtime', 'releases')

If the '*Attack*' or '*Release*' method discovers that some attack/release phases are overlapped, the redundant onset is filtered out. If for instance, two successive peaks have either the same start attack time, or the same end release time, only one peak is kept.

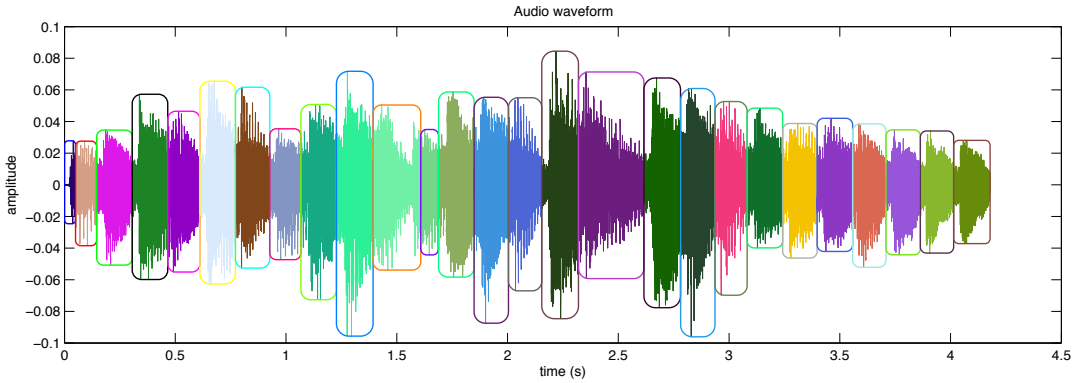
Segmentation

The onset points can be used for segmentation of the initial waveform:



o = mironsets('ragtime'); mirsegment('ragtime', o)

Alternatively, the beginning of the attack phases can be used for the segmentation:



o = mironsets('ragtime', 'Attacks'); mirsegment('ragtime', o)

Frame decomposition

If the onset detection curve is not a scalar object (i.e., basically, if the output is an envelope), it can be further decomposed into frames if the '*Frame*' option has been specified, with default frame length 3 seconds and hop factor .1

Preselected Model

Complete (or nearly complete) models are available:

- *mironsets(..., '**Scheirer**')* follows the model proposed in (Scheirer, 1998). Il corresponds to

mironsets(..., 'FilterbankType', 'Scheirer', 'FilterType', 'HalfHann', 'Sampling', 200, 'HalfwaveDiff', 'Sum', 0, 'Detect', 0)

- *mirenvelope(..., **'Klapuri99')*** follows the model proposed in (Klapuri., 1999). Il corresponds to

o = mironsets(..., 'FilterbankType', 'Klapuri', 'FilterType', 'HalfHann', 'PreDecim', 180, 'Sum', 0, 'PostDecim', 0);

o2= mirenvelope(o, 'HalfwaveDiff'); % absolute distance function D

o= mirenvelope(o, 'Mu', 'HalfwaveDiff'); % relative distance function W

p = mirpeaks(o, 'Contrast', .2, 'Chrono');

p2 = mirpeaks(o2, 'ScanForward', p, 'Chrono');

o = combinepeaks(p, p2, .05);

where *combinepeaks* is a dedicated function that creates a curve made of burst at position of peaks *p* and with amplitude related to peaks *p2*.

o = mirsum(o, 'Weights', fB);

The intensity is multiplied by the band center frequency *fB*.

o = mirenvelope(o, 'Smooth', 12);

Accessible Output

cf. §5.2 for an explanation of the use of the *get* method. Specific fields:

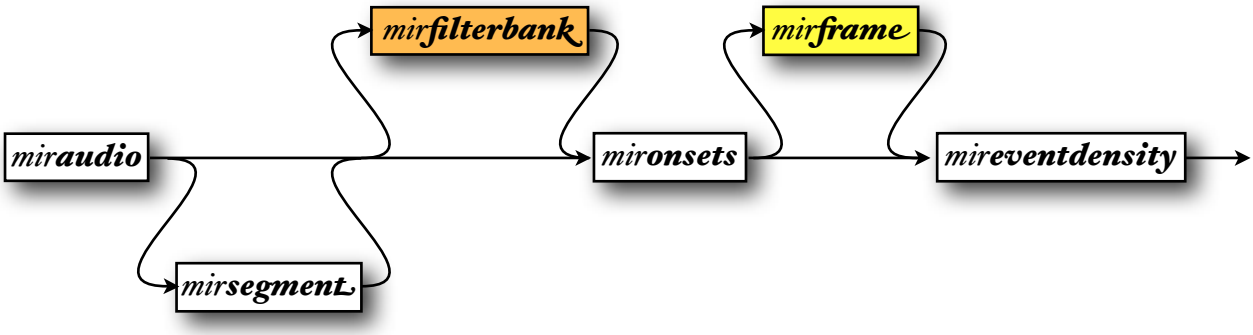
- '***AttackPos***': the abscissae position of the starting attack phases, in sample index,
- '***AttackPosUnit***': the abscissae position of the starting attack phases, in the default abscissae representation,
- '***ReleasePos***': the abscissae position of the ending release phases, in sample index,
- '***ReleasePosUnit***': the abscissae position of the ending release phases, in the default abscissae representation.

mireventdensity

Description

Estimates the average frequency of events, i.e., the number of note onsets per second.

Flowchart Interconnections

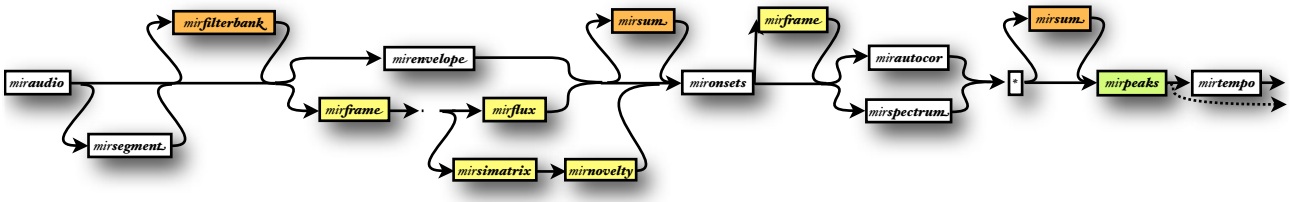


mirtempo

Description

Estimates the tempo by detecting periodicities from the onset detection curve.

Flowchart Interconnections



The tempo can be estimated in various ways:

- *mirtempo(..., '**Autocor**')* computes an autocorrelation function of the onset detection curve, using ***mirautocor*** (default choice). Options related to *mirautocor* can be specified:
  - '***Enhanced***' (toggled on by default[3] here),
  - '***Resonance***' (set by default to '*ToiviainenSnyder*'),
  - '***NormalWindow***' (same default value).
- *mirtempo(..., '**Spectrum**')* computes a spectral decomposition of the onset detection curve, using ***mirspectrum***. Options related to *mirspectum* can be passed here as well:
  - '***ZeroPad***' (set by default here to 10 000 samples),
  - '***Prod***' (same default, when toggled on, as for *mirspectrum*),
  - '***Resonance***' either '*ToiviainenSnyder*' (default value) or 0, '*off*', or '*no*'.
- *mirtempo(..., '**Autocor**', '**Spectrum**')* combines both strategies: the autocorrelation function is translated into the frequency domain in order to be compared to the spectrum curve, and the two curves are subsequently multiplied.

Then a peak picking is applied to the autocorrelation function or to the spectrum representation. The parameters of the peak picking can be tuned.

- *mirtempo(..., '**Total**', m)* selects not only the best tempo, but the *m* best tempos.
- *mirtempo(..., '**Min**', mi)* indicates the lowest tempo taken into consideration, expressed in bpm. Default value: 40 bpm.

---

[3] except when '*Track*' option is used, as explained.

- *mirtempo(..., 'Max', ma)* indicates the highest tempo taken into consideration, expressed in bpm. Default value: 200 bpm.
- *mirtempo(..., 'Track', t)* tracks peaks along time, in order to obtain a stabilized tempo curve and to limit therefore switches between alternative pulsations. Default value when option toggled on: t = 0.1 s. When '*Track*' is toggled on, '*Enhanced*' is forced to off.

When *'Track'* is used for academic research, please cite the following publication:

Olivier Lartillot, "*mirtempo*: Tempo estimation through advanced frame-by-frame peaks tracking", *Music Information Retrieval Evaluation eXchange (MIREX 2010)*.

- *mirtempo(..., 'Contrast', c)* specifies the contrast factor for the peak picking. Default value: c = 0.1
- *mirtempo(..., 'Nearest', n)* chooses the peak closest to n (in s.). Default value when option toggled on: n = 0.5 s.

mirtempo accepts as input data type either:

- *mirautocor* objects,
- *mirspectrum* objects,
- onset detection curve (resulting from *mironsets*), frame-decomposed or not, channel-decomposed or not,
- and all the input data accepted by *mironsets*.

The onset detection curve computed in *mironsets* can be controlled using the following options:

- '***Envelope***' (default) and '***DiffEnvelope***':
  - with the '***Method***' set by default to '***Filter***':
    - with '***FilterType***' option with same default,
    - with '***Filterbank***' option set to 10 by default,
    - with '***FilterbankType***' option with same default,
  - '*Method*' can be set to '*Spectro*' as well, and the *'**Freq**','**Mel**','**Bark**','**Cents**'* selection can be specified, with same default.

- Besides '*Method*': '***HalfwaveCenter***', '***HalfwaveDiff***', '***Lambda***', '***Center***', '***Smooth***', '***Sampling***', '***Log***' and '***Mu***', all with same default and '***Diff***' set to '*On*' by default.

- '***SpectralFlux***': with '***Complex***', '***Inc***', '***Median***' and '***Halfwave***' with same default.
- and '***Pitch***'.

Other options related to *mironsets* can be specified:

- '***Filterbank***', with same default value than for *mironsets,*
- '***Frame***', with same default value than for *mironsets*,
- *mironsets(..., '**Sum**', w)* specifies when to sum the channels. Possible values:
  - *w* = '*Before*': sum before the autocorrelation or spectrum computation.
  - *w* = '*After*': autocorrelation or spectrum computed for each band, and summed into a "summary".
  - *w* = 0: tempo estimated for each band separately, with no channel recombination.

mirtempo can return several outputs:

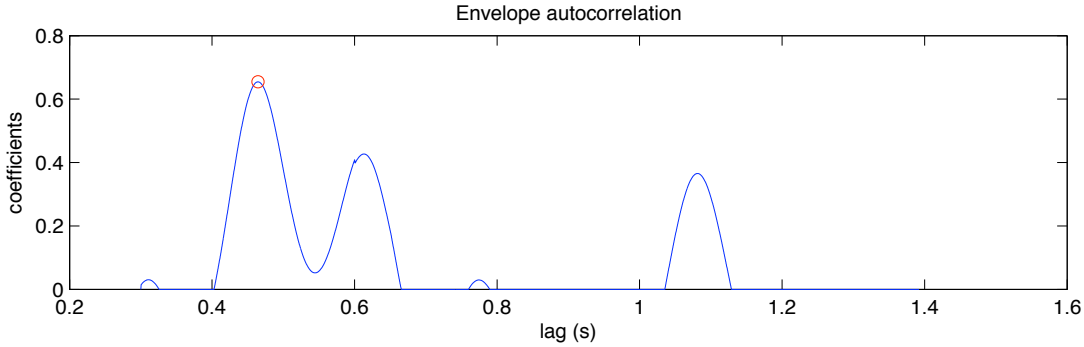
1. the tempo itself (or set of tempi) and
2. the ***mirspectrum*** or ***mirautocor*** data, where is highlighted the (set of) peak(s) corresponding to the estimated tempo (or set of tempi).

EXAMPLE

The tempo estimation related to the *ragtime* example

[t ac] = mirtempo('ragtime')

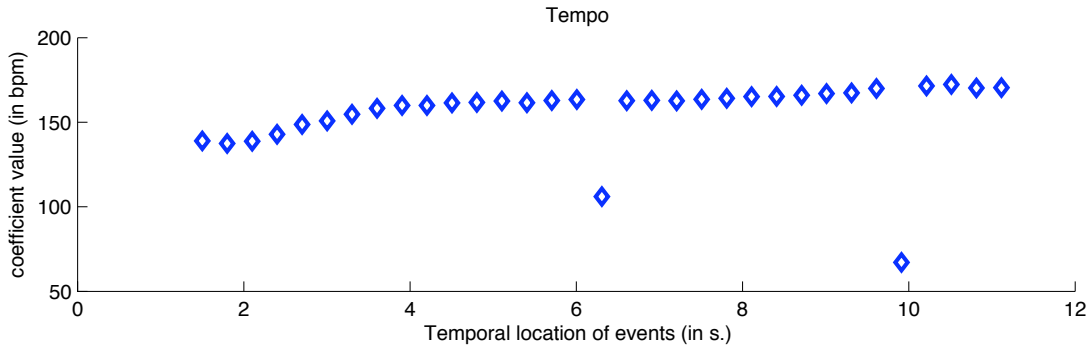
leads to a tempo *t* = 129.1832 bpm and to the following autocorrelation curve *ac*:



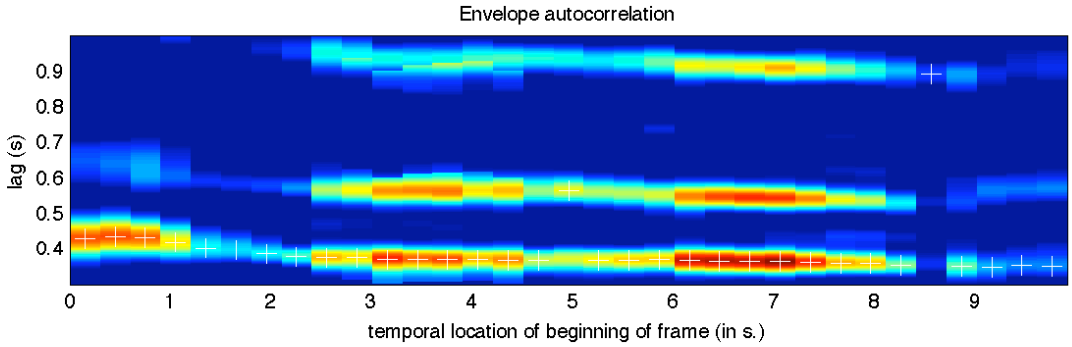
The frame-decomposed tempo estimation related to the *czardas* example

[t ac] = mirtempo('czardas', 'Frame')

leads to the following tempo curve *t*:



and the following autocorrelation frame decomposition *ac*:



mirpulseclarity

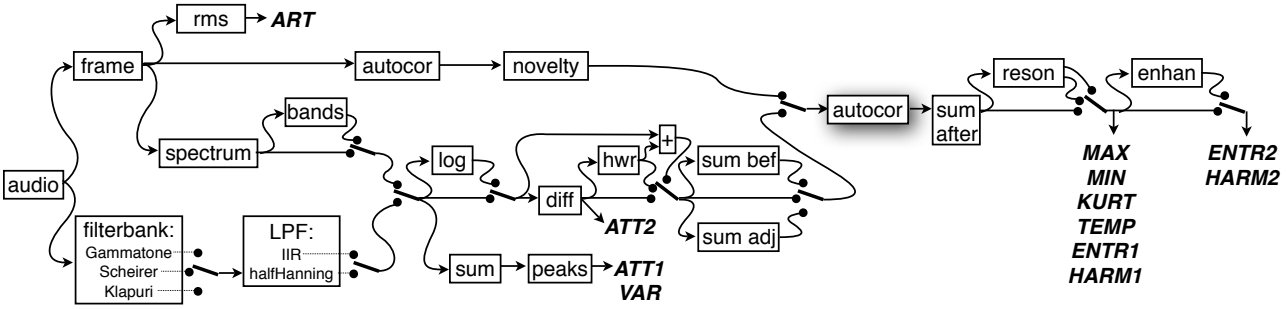
When *mirpulseclarity* is used for academic research, please cite the following publication:

Olivier Lartillot, Tuomas Eerola, Petri Toiviainen, Jose Fornari, "Multi-feature modeling of pulse clarity: Design, validation, and optimization", *International Conference on Music Information Retrieval*, Philadelphia, 2008.

Description

Estimates the rhythmic clarity, indicating the strength of the beats estimated by the *mirtempo* function.

Flowchart Interconnections



The pulse clarity can be estimated in various ways:

- *mirpulseclarity(..., s)* selects a particular heuristic for pulse clarity estimation. Most heuristics are based on the autocorrelation curve computed for tempo estimation (i.e., the second output of *mirtempo*) (Lartillot, Eerola, Toiviainen, and Fornari, 2008):
  - *s* = '***MaxAutocor***' selects the maximum correlation value in the autocorrelation curve (default heuristic).
  - *s* = '***MinAutocor***' selects the minimum correlation value in the autocorrelation curve.
  - *s* = '***MeanPeaksAutocor***' averages the local maxima in the autocorrelation curve.
  - *s* = '***KurtosisAutocor***' computes the kurtosis of the autocorrelation curve.
  - *s* = '***EntropyAutocor***' computes the entropy of the autocorrelation curve.
  - *s* = '***InterfAutocor***' computes the harmonic relations between pulsations.
  - *s* = '***TempoAutocor***' selects the tempo related to the highest autocorrelation.

Others heuristics are based more simply on the onset curve itself:

- *s* = '***Articulation***' estimates the average silence ratio of the onset curve (option '*ASR*' in *mirlowenergy*.
- *s* = '***Attack***' averages the attack slopes of all onsets (the '*Diff*', '*Gauss*' can be specified, with same default).
- *s* = '***ExtremEnvelope***' estimates the total amplitude variability of the onset curve.

*mirpulseclarity(..., '**Model**', m)* selects one out of two possible models that have been found as optimal in our experiments (Lartillot, Eerola, Toiviainen, and Fornari, 2008):

- *m* = 1 selects the default model with its associated weight.
- *m* = 2 selects the following model: '*Gammatone*', no log, no '*Resonance*', '*Lambda*' set to .8, and '*Sum*' set to '*After*', with its associated weight.
- *m* = [1 2] sums the two models altogether.

The onset detection curve computed in *mironsets* can be controlled using the following options:

- '***Envelope***' (default) and '***DiffEnvelope***':
  - with the '***Method***' set by default to '***Spectro***', and the ***'Freq','Mel','Bark','Cents'*** selection can be specified, with same default.
  - '*Method*' can be set to '*Filter*' as well:
    - with '***FilterType***' option with same default,
    - with '***Filterbank***' option set to 20 by default,
    - with '***FilterbankType***' option set to '*Scheirer*' by default,
  - Besides '*Method*': '***HalfwaveDiff***', '***Lambda***', '***Smooth***', '***Log***' with same default, and '***Mu***', set by default here to 100.
- '***SpectralFlux***': with '***Inc***' with same default, and '***Median***' and '***Halfwave***' toggled off by default.
- and '***Pitch***'.

The autocorrelation function performed in *mirautocor* can be controlled using the following options:

- '***Enhanced***' (toggled off by default; forced to '*Off*' in '*MinAutocor*'),
- '***Resonance***', '***Min***', '***Max***' (with same default as in *mirautocor*).

Some further options operates as in *mirtempo*:

- '***Sum***',
- '***Total***' (ignored in '*MaxAutocor*', '*MinAutocor*' and '*EntropyAutocor*' methods),
- '***Contrast***': with a default value set to .01,
- '***Frame***': if specified, set by default to 5 s and 10% hop.

mirpulseclarity accepts as input data type either:

- *mirautocor* objects,
- onset detection curve (resulting from *mironsets*), frame-decomposed or not, channel-decomposed or not,
- and all the input data accepted by *mironsets*.

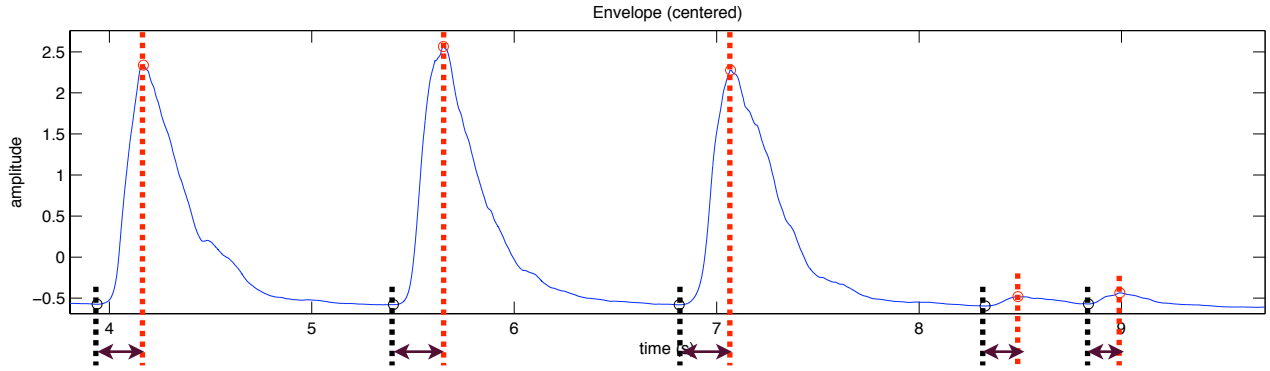
mirpulseclarity can return several outputs:

1. the pulse clarity value and
2. the ***mirautocor*** data that was used for the estimation of pulse clarity.

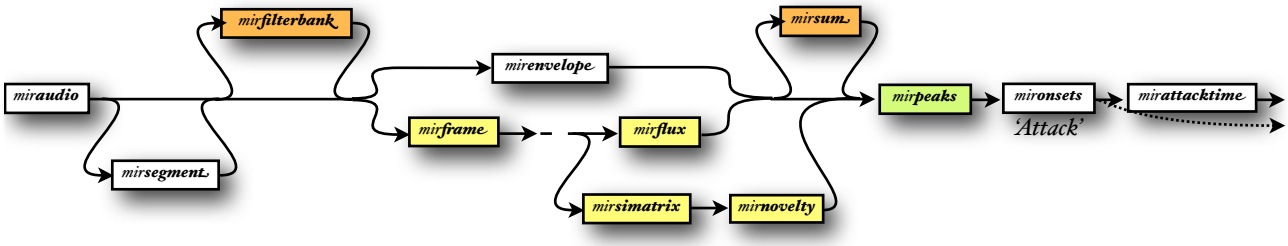
mirattacktime

Description

The attack phase detected using the '*Attacks*' option in *mironsets* can offer some timbral characterizations. One simple way of describing the attack phase, proposed in *mirattacktime*, consists in estimating its temporal duration.



Flowchart Interconnections



mirattacktime accepts as input data type either:

- onset detection curves (resulting from ***mironsets***), already including peaks or not,
- and all the input data accepted by *mironsets*.

mirattacktime can return several outputs:

1. the attack time itself and
2. the onset detection curve returned by ***mironsets***, including the detected onsets.

Options

- *mirattacktime(..., **'Lin'**)* returns the duration in a linear scale (in seconds). (Default choice)

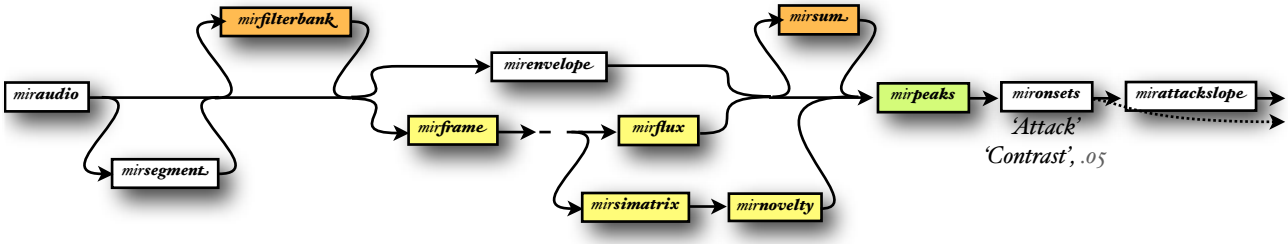
- *mirattacktime(..., '**Log**')* returns the duration in a log scale (Krimphoff et al., 1994).

mirattackslope

Description

Another description of the attack phase is related to its average slope.

Flowchart Interconnections



mirattackslope accepts as input data type either:

- onset detection curves (resulting from ***mironsets***),
- and all the input data accepted by *mironsets*.

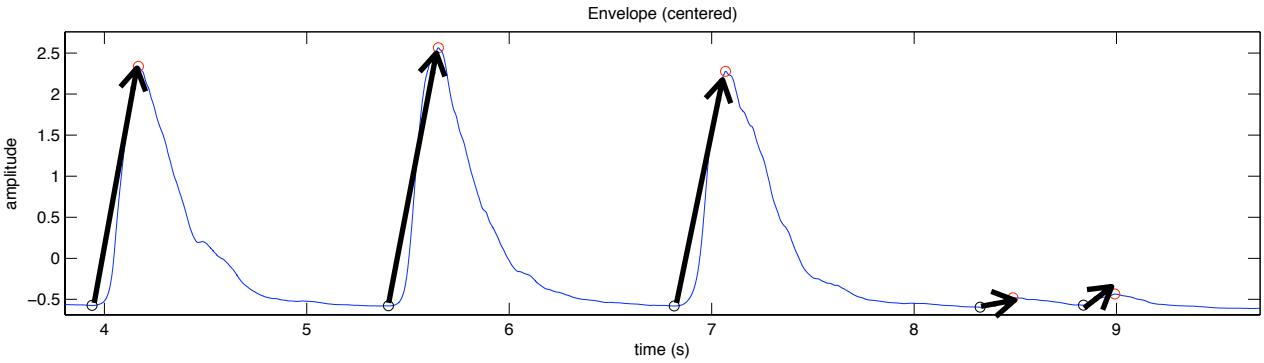
The peak picking from the onset detection is performed in any case. Its *'Contrast'* parameter can be specified. Its default value is the same as in *mironsets*.

mirattackslope can return several outputs:

1. the attack slope itself and
2. the onset detection curve returned by ***mironsets***, including the detected onsets.

Options

- *mirattackslope(x, meth)* specifies the method for slope estimation. Possible values for *meth* are:
  - *meth* = '***Diff***' computes the slope as a ratio between the magnitude difference at the beginning and the ending of the attack period, and the corresponding time difference. (Default choice)

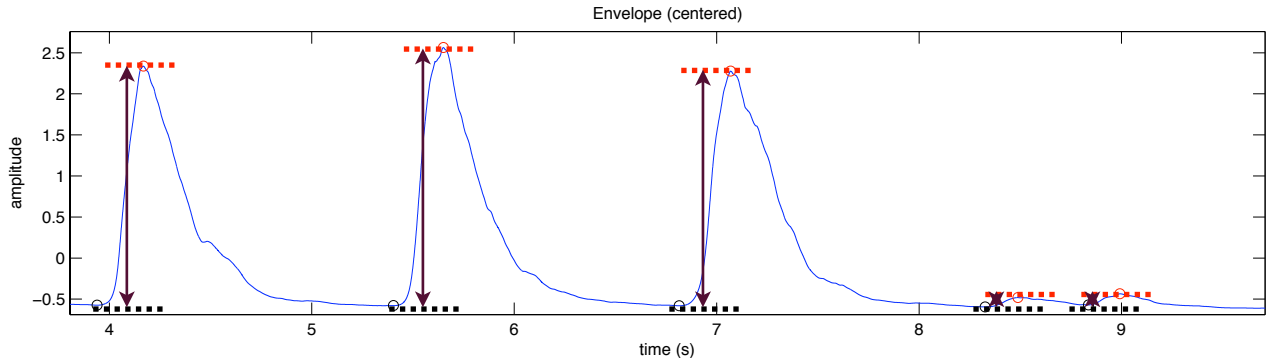


- *meth* = '***Gauss***' computes the average of the slope, weighted by a gaussian curve that emphasizes values at the middle of the attack period (similar to Peeters, 2004).

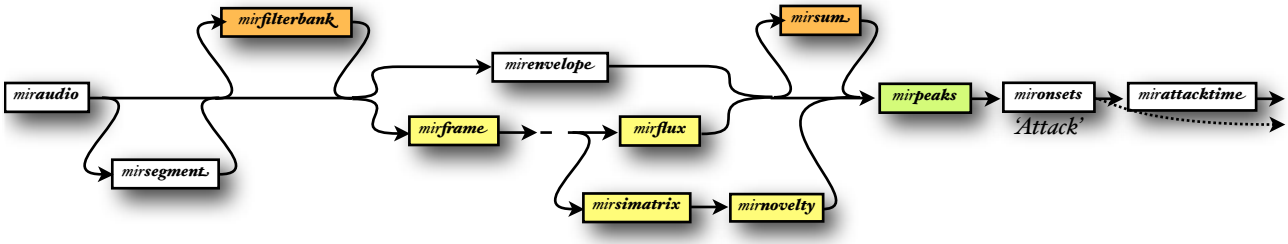
mirattackleap

DESCRIPTION

Another simple way of describing the attack phase, proposed in *mirattackleap*, consists in estimating the amplitude difference between the beginning and the end of the attack phase.



FLOWCHART INTERCONNECTIONS



mirattackleap accepts as input data type either:

- onset detection curves (resulting from ***mironsets***), already including peaks or not,
- and all the input data accepted by *mironsets*.

The peak picking from the onset detection is performed in any case. Its *'Contrast'* parameter can be specified. Its default value is the same as in *mironsets*.

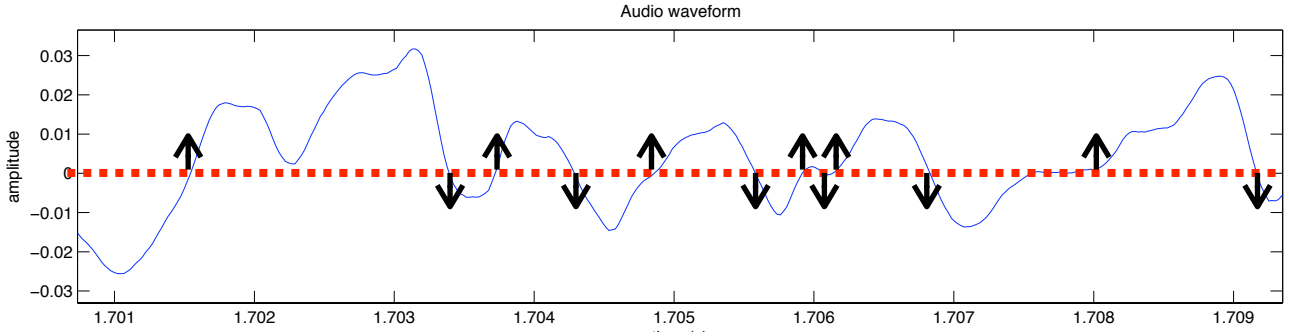
mirattackleap can return several outputs:

1. the attack leap itself and
2. the onset detection curve returned by ***mironsets***, including the detected onsets.

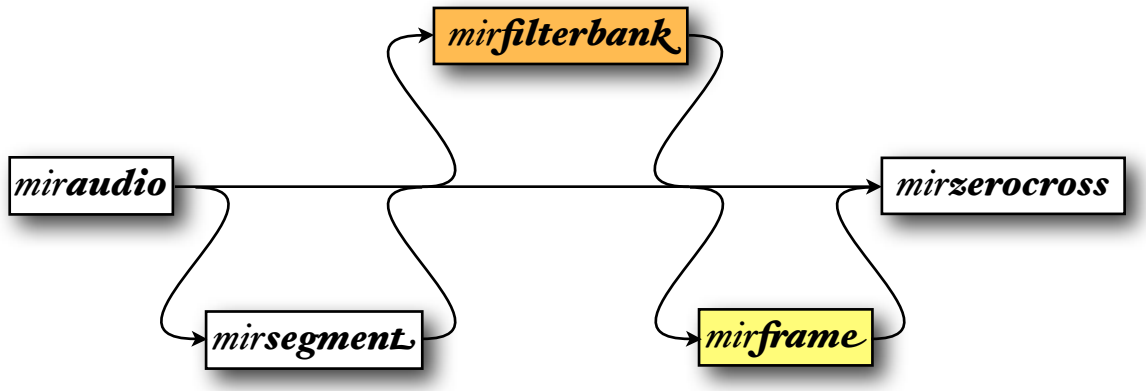
mirzerocross

Waveform sign-change rate

A simple indicator of noisiness consists in counting the number of times the signal crosses the X-axis (or, in other words, changes sign).



Flowchart Interconnections



mirzerocross actually accepts any input data type (cf. section 4.2).

Options

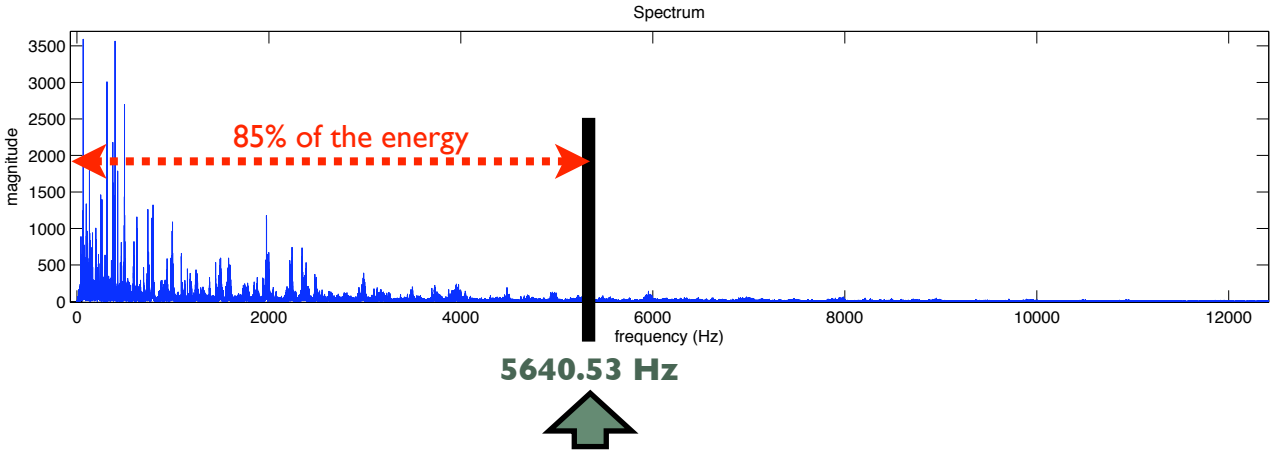
- *mirzerocross(..., 'Per', p)* precises the temporal reference for the rate computation. Possible values:
  - *p* = *'Second'*: number of sign-changes per second (Default).
  - *p* = *'Sample'*: number of sign-changes divided by the total number of samples. The *'Second'* option returns a result equal to the one returned by the *'Sample'* option multiplied by the sampling rate.
- *mirzerocross(..., 'Dir', d)* precises the definition of sign change. Possible values:
  - *d* = *'One'*: number of sign-changes from negative to positive only (or, equivalently, from positive to negative only). (Default)

- *d = 'Both'*: number of sign-changes in both ways. The '*Both*' option returns a result equal to twice the one returned by the '*One*' option.

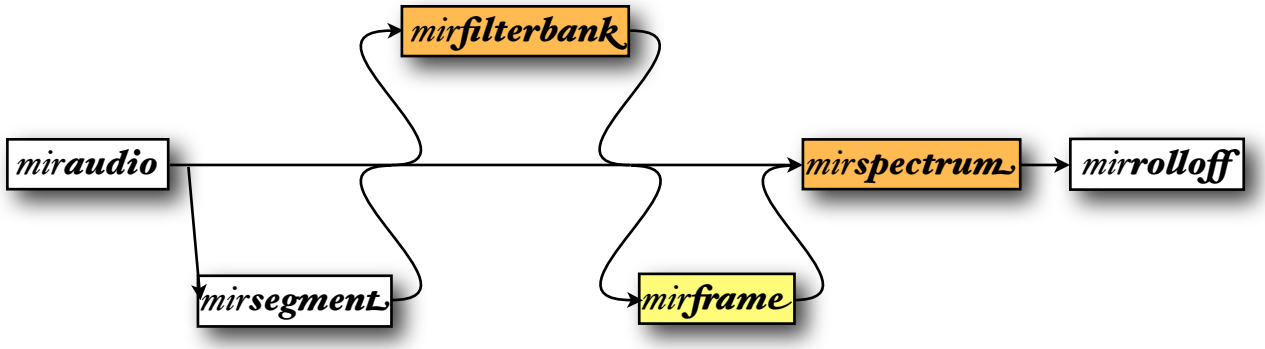
mirrolloff

High-frequency energy (I)

One way to estimate the amount of high frequency in the signal consists in finding the frequency such that a certain fraction of the total energy is contained below that frequency. This ratio is fixed by default to .85 (following Tzanetakis and Cook, 2002), other have proposed .95 (Pohle, Pampalk and Widmer, 2005).



Flowchart Interconnections



mirrolloff accepts either:

- ***mirspectrum*** objects, or
- ***miraudio*** objects (same as for *mirspectrum*),
- **file name** or the '***Folder***' keyword.

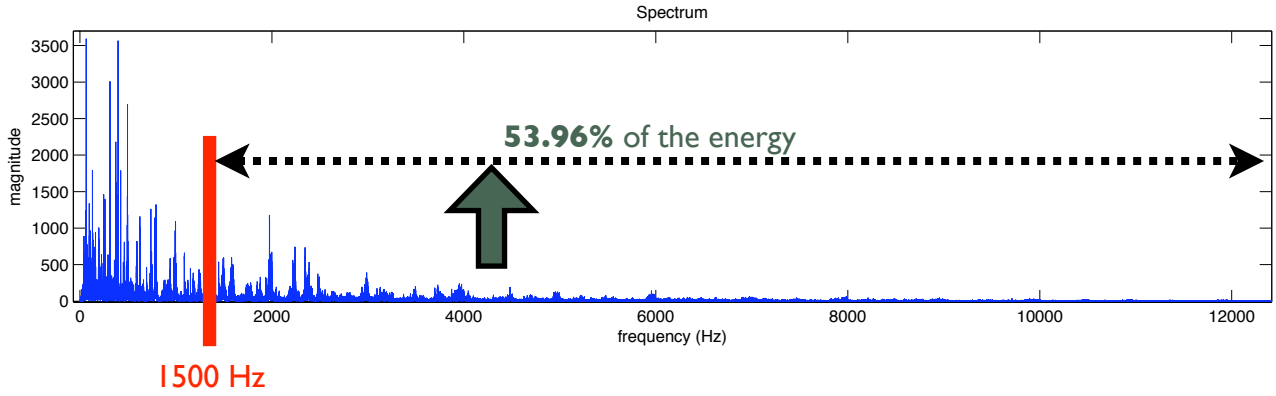
Option

*mirrolloff(..., '**Threshold**', p)* specifies the energy threshold, as a percentage. Default value: .85

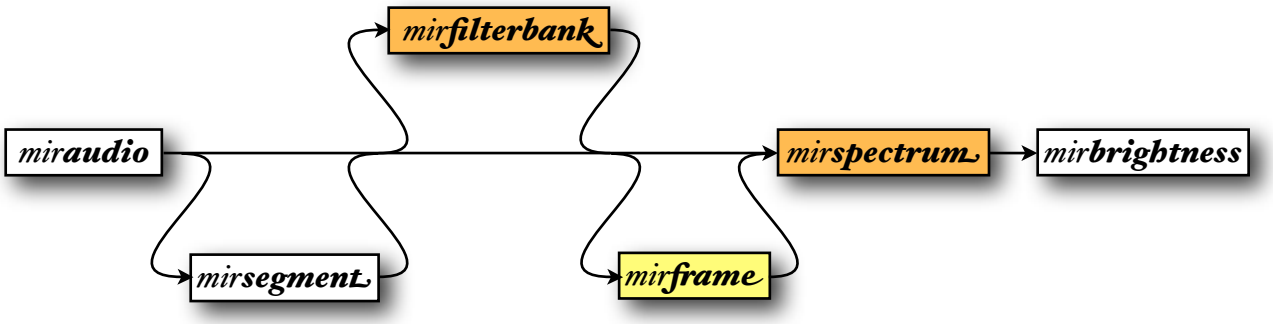
mirbrightness

High-frequency energy (II)

A dual method consists in fixing this time the cut-off frequency, and measuring the amount of energy above that frequency (Juslin, 2000). The result is expressed as a number between 0 and 1.



Flowchart Interconnections



mirbrightness accepts either:

- ***mirspectrum*** objects, or
- ***miraudio*** objects (same as for *mirspectrum*).
- **file name** or the '***Folder***' keyword.

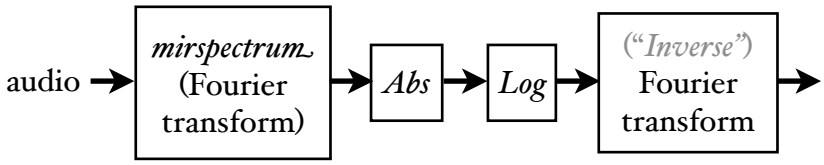
Options

*mirbrightness(..., '**CutOff**', f)* specifies the frequency cut-off, in Hz. Default value: 1500 Hz. The value 1000 Hz has been proposed in (Laukka, Juslin and Bresin, 2005), and the value of 3000 Hz has been proposed in (Juslin, 2000).

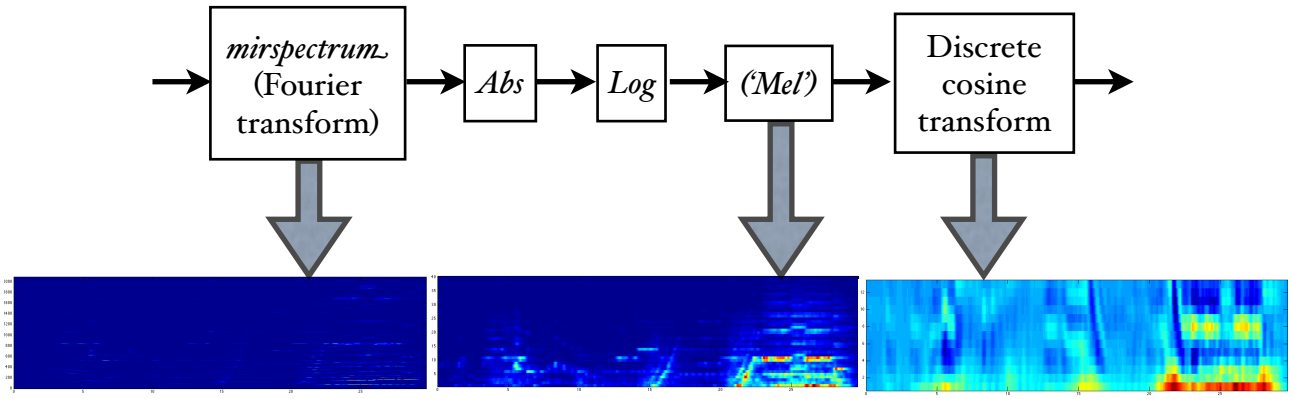
mirmfcc

Mel-Frequency Cepstral Coefficients

MFCC offers a description of the spectral shape of the sound. We recall that the computation of the cepstrum followed the following scheme:



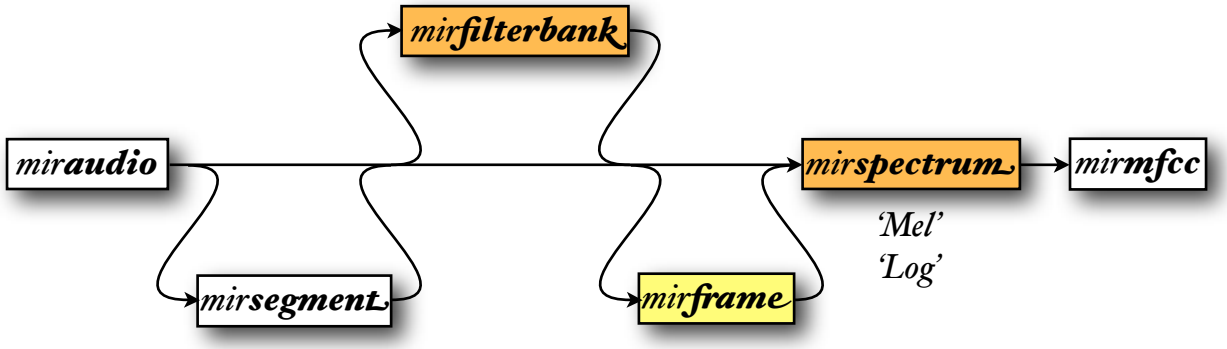
The computation of mel-frequency cepstral coefficients is highly similar:



Here the frequency bands are positioned logarithmically (on the Mel scale) which approximates the human auditory system's response more closely than the linearly-spaced frequency bands. And the Fourier Transform is replaced by a Discrete Cosine Transform. A discrete cosine transform (DCT) is a Fourier-related transform similar to the discrete Fourier transform (DFT), but using only real numbers. It has a strong "energy compaction" property: most of the signal information tends to be concentrated in a few low-frequency components of the DCT. That is why by default only the first 13 components are returned.

By convention, the coefficient of rank *zero* simply indicates the average energy of the signal.

Flowchart Interconnections



mirmfcc accepts either:

- ***mirspectrum*** objects, or
- ***miraudio*** objects (same as for *mirspectrum*),
- **file name** or the '***Folder***' keyword.

mirmfcc can return several outputs:

- the mfcc coefficients themselves and
- the spectral representation (output of *mirspectrum*), in mel-band and log-scale.

Options

- *mirmfcc(..., '**Bands**', b)* indicates the number of bands used in the mel-band spectrum decomposition. By default, *b* = 40.
- *mirmfcc(..., '**Rank**', N)* computes the coefficients of rank(s) *N*. The default value is *N* = 1:13. Beware that the coefficient related to the average energy is by convention here of rank 0. This zero value can be included to the array *N* as well.
- If the output is frame-decomposed, showing the the temporal evolution of the MFCC along the successive frames, the temporal differentiation can be computed:
  - *mirmfcc(..., '**Delta**', d)* performs temporal differentiations of order *d* of the coefficients, also called delta-MFCC (for *d* = 1) or delta-delta-MFCC (for *d* = 2). By default, *d* = 1.
  - *mirmfcc(..., '**Radius**', r)* specifies, for each frame, the number of successive and previous neighbouring frames taken into consideration for the least-square approximation used for the derivation. For a given radius r, the Delta operation for each frame i is computed by summing the MFCC coefficients at frame i+j (with j from -r to +r) , each coefficient being multiplied by its weight j. Usually the radius is equal to 1 or 2. Default value: *r* = 2.

Accessible Output

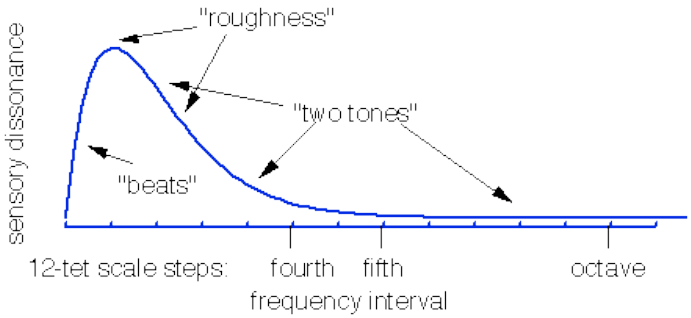
cf. §5.2 for an explanation of the use of the *get* method. Specific fields:

- '***Rank***': the series of rank(s) taken into consideration (same as '*Pos*'),
- '***Delta***': the number of times the delta operation has been performed.

mirroughness

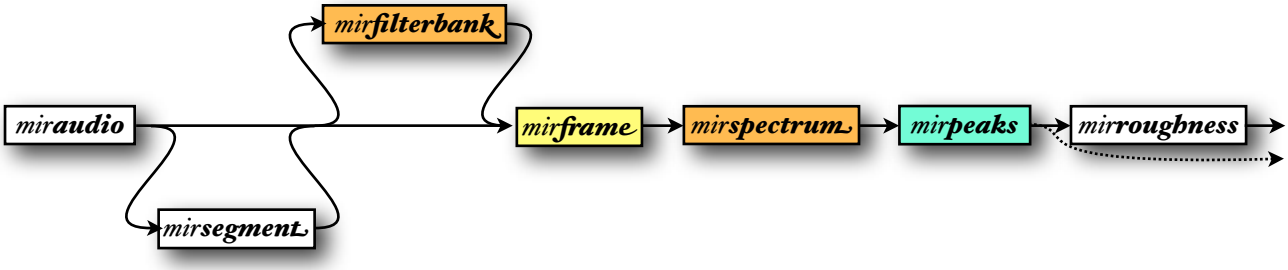
Sensory dissonance

Plomp and Levelt (1965) has proposed an estimation of the sensory dissonance, or roughness, related to the beating phenomenon whenever pair of sinusoids are closed in frequency. The authors propose as a result an estimation of roughness depending on the frequency ratio of each pair of sinusoids represented as follows:



An estimation of the total roughness is available in *mirroughness* by computing the peaks of the spectrum, and taking the average of all the dissonance between all possible pairs of peaks (Sethares, 1998).

Flowchart Interconnections



The '***Contrast***' parameter associated to ***mirpeaks*** can be specified, and is set by default to .01

mirroughness accepts either:

- ***mirspectrum*** objects, where peaks have already been picked or not,
- ***miraudio*** objects: same as for *mirspectrum*, except that a **frame decomposition** is automatically performed, with default frame length 50 ms and half overlapping. This default frame decomposition is due to the fact that roughness can only be associated to a spectral representation association to a short-term sound excerpt: there is no sensory dissonance provoked by a pair of sinusoid significantly distant in time.
- **file name** or the '***Folder***' keyword.

mirroughness can return several outputs:

1. the roughness value itself and
2. the spectral representation (output of ***mirspectrum***) showing the picked peaks (returned by ***mirpeaks***).

Options

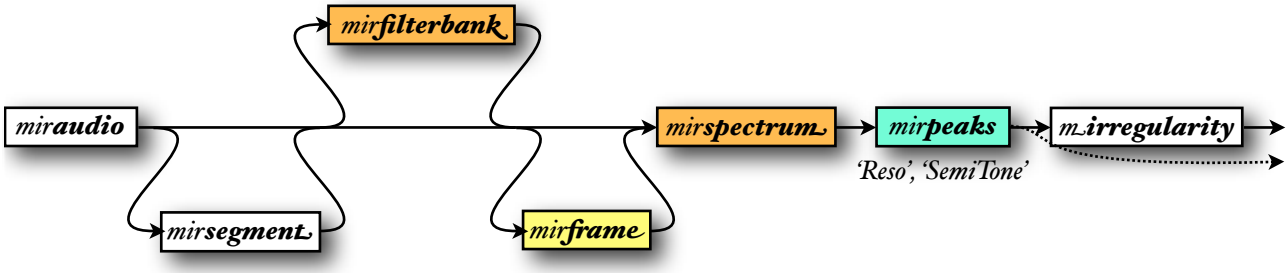
- *mirroughness(..., m)* specifies the method used:
  - *m* = '***Sethares***' (default): based on the summation of roughness between all pairs of sines (obtained through spectral peak-picking) (Sethares, 1998). For each pair of peaks, the corresponding elementary roughness is obtained by multiplying the two peak amplitudes altogether, and by weighting the results with the corresponding factor given on the dissonance curve.
  - *m* = '***Vassilakis***': variant of '*Sethares*' model with a more complex weighting (Vassilakis, 2001, Eq. 6.23).

mirirregularity

Spectral peaks variability

The irregularity of a spectrum is the degree of variation of the successive peaks of the spectrum.

Flowchart Interconnections



The '**Contrast**' parameter associated to **mirpeaks** can be specified, and is set by default to .01

mirregularity accepts either:

- **mirspectrum** objects, where peaks have already been picked or not,
- **miraudio** objects (same as for *mirspectrum*),
- **file name** or the '**Folder**' keyword.

Options

- *mirregularity*(..., '**Jensen**') is based on (Jensen, 1999), where the irregularity is the sum of the square of the difference in amplitude between adjoining partials.(Default approach)

$$\left(\sum_{k=1}^{N}(a_k - a_{k+1})^2\right) / \sum_{k=1}^{N} a_k^2$$

- *mirregularity*(..., '**Krimphoff**') is based on (Krimphoff et al., 1994), where the irregularity is the sum of the amplitude minus the mean of the preceding, same and next amplitude.

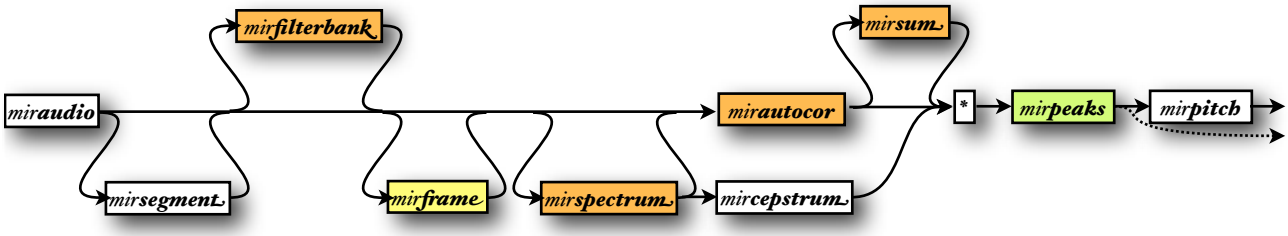
$$\sum_{k=2}^{N-1}\left|a_k - \frac{a_{k-1} + a_k + a_{k+1}}{3}\right|$$

*mir**pitch***

Pitch estimation

Extract pitches, returned either as continuous pitch curves or as discretized note events.

Flowchart Interconnections



The pitch content can be estimated in various ways:

- *mirpitch(..., '**Autocor**')* computes an autocorrelation function of the audio waveform, using ***mirautocor***. This is the default strategy. Options related to *mirautocor* can be specified:
  - '***Enhanced***', toggled on by default here,
  - '***Compress***', set by default to .5,
  - filterbank configuration can be specified: either *'**2Channels**'* (default configuration), *'**Gammatone**' or '**NoFilterbank**'*,
  - if a filterbank is used, '***Sum***' specifies whether the channels are recombined once the autocorrelation function is computed (*'Sum', 1*, which is the default), or if on the contrary, the channels are kept separate, and pitch content is extracted in each channel separately (*'Sum', 0*).
- *mirpitch(..., '**Spectrum**')* computes the FFT spectrum (***mirspectrum***).
- *mirpitch(..., '**AutocorSpectrum**')* computes the autocorrelation (***mirautocor***) of the FFT spectrum (***mirspectrum***).
- *mirpitch(..., '**Cepstrum**')* computes the cepstrum (***mircepstrum***).
- These methods can be combined. In this case, the resulting representations (autocorrelation function or cepstral representations) are all expressed in the frequency domain and multiplied altogether.

Then a peak picking is applied to the autocorrelation function or to the cepstral representation. The parameters of the peak picking can be tuned.

- *mirpitch(..., '**Total**', m)* selects only the *m* best pitches.
- *mirpitch(..., '**Mono**')* only select the best pitch, corresponding hence to *mirpitch(..., '**Total**', 1)*.
- *mirpitch(..., '**Min**', mi)* indicates the lowest pitch taken into consideration, in Hz. Default value: 75 Hz, following a convention in the *Praat* software (Boersma & Weenink, 2005).
- *mirpitch(..., '**Max**', ma)* indicates the highest pitch taken into consideration, expressed in Hz. Default value: 2400 Hz, because there seem to be some problems with higher frequency, due probably to the absence of pre-whitening in our implementation of Tolonen and Karjalainen autocorrelation approach (used by default).
- *mirpitch(..., '**Threshold**', t)* specifies the threshold factor for the peak picking. Default value: *c* = 0.4.
- *mirpitch(..., '**Contrast**', c)* specifies the contrast factor for the peak picking. Default value: *c* = 0.1.
- *mirpitch(..., '**Order**', o)* specifies the ordering for the peak picking. Default value: *o* = '*Amplitude*'.

mirpitch accepts as input data type either:

- ***mirpitch*** objects,
- output of ***mirpeaks*** computation,
- ***mirautocor*** objects,
- ***mircepstrum*** objects,
- ***mirspectrum*** objects,
- ***miraudio*** objects, where the audio waveform can be:
  - segmented (using ***mirsegment***),
  - when pitch is estimated by autocorrelating the audio waveform ('*Autocor*' strategy), the audio waveform is be default first decomposed into channels (cf. the '***Filterbank***' option below),

- decomposed into frames or not, using ***mirframe***, or the '***Frame***' option, where the default frame length is .46.4 ms and the default hop length is 10 ms (Tolonen & Karjalainen 2000);

- **file name** or the '***Folder***' keyword: same behavior than for *miraudio* objects.

mirpitch can return several outputs:

1. the pitch frequencies themselves, and

2. the ***mirautocor*** or ***mircepstrum*** data, where is highlighted the (set of) peak(s) corresponding to the estimated pitch (or set of pitches).

Post-Processing Options

- *mirpitch(..., '**Cent**')* convert the pitch axis from Hz to cent scale. One octave corresponds to 1200 cents, so that 100 cents correspond to a semitone in equal temperament.

- *mirpitch(..., '**Segment**')* segments the obtained monodic[4] pitch curve in cents as a succession of notes with stable frequencies.

When *'Segment'* is used for academic research, please cite the following publication:

> Olivier Lartillot, "Computational analysis of maqam music: From audio transcription to musicological analysis, everything is tightly intertwined", *Acoustics 2012 Hong Kong Conference*.

- *mirpitch(..., '**Median**', l)* performs a median filtering of the pitch curve. The length of the median filter is given by *l* (in s.). Its default value is .1 s. The median filtering can only be applied to mono-pitch curve. If several pitches were extracted in each frame, a mono-pitch curve is first computed by selecting the best peak of each successive frame.

- *mirpitch(..., '**Stable**', th, n)* remove pitch values when the difference (or more precisely absolute logarithmic quotient) with the *n* precedent frames exceeds the threshold *th*.

  - if *th* is not specified, the default value .1 is used.

  - if *n* is not specified, the default value 3 is used.

- *mirpitch(..., '**Reso**', '**SemiTone**')* removes peaks whose distance to one or several higher peaks is lower than a given threshold 2^(1/12) (corresponding to a semitone).

---

[4] This '*Segment*' option requires a monodic pitch curve extraction using the '*Mono*' option, which is therefore toggled on, as well as the '*Cent*' and '*Frame*' options.

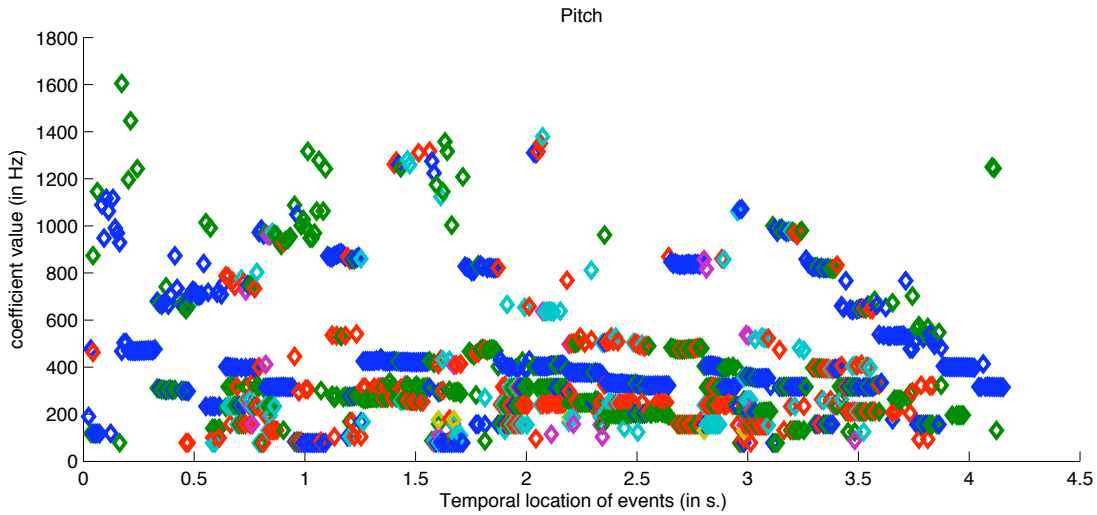
Preset Model

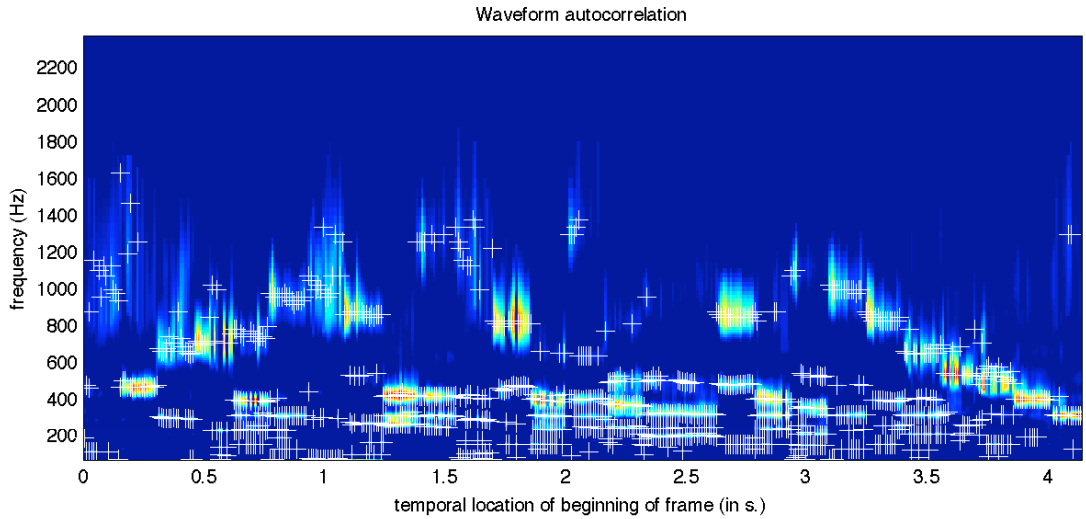
- *mirpitch(..., '**Tolonen**')* implements (part of) the model proposed in (Tolonen & Karjalainen, 2000). It is equivalent to

mirpitch(..., 'Enhanced', 2:10, 'Generalized', .67, '2Channels')

Example

[p ac] = mirpitch('ragtime', 'Frame')





Accessible Output

cf. §5.2 for an explanation of the use of the *get* method. Specific fields:

- '***Amplitude***': the amplitude associated with each pitch component.

Importation of Pitch Data

mirpitch(f, a, r) creates a *mirpitch* object based on the frequencies specified in *f* and the related amplitudes specified in *a*, using a frame sampling rate of *r* Hz (set by default to 100 Hz).

Both *f* and *a* can be either:

- a matrix where each column represent one pitch track and lines corresponds to frames,
- an array of cells, where each cell, representing one individual frame, contains a vector.

mirmidi

Automated Transcription

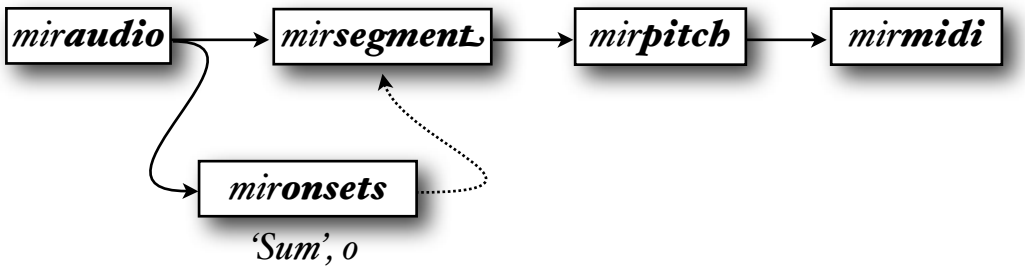
Segments the audio into events, extracts pitches related to each event and attempts a conversion of the result into a MIDI representation.

The audio segmentation is based on the onset detection given by *mironsets* with the default parameter, but with the '*Sum*' option toggled off in order to keep the channel decomposition of the input audio data.

The MIDI output is represented using the *MIDI Toolbox* note matrix representation. The displayed output is the piano-roll representation of the MIDI data, which requires *MIDI Toolbox*. Similarly, the result can be:

- sonified using ***mirplay***, with the help of *MIDI Toolbox;*
- saved using ***mirsave***:
  - as a MIDI file (by default), with the help of *MIDI Toolbox*,
  - as a *LilyPond* file (if the specified file has a '*.ly*' extension).

Flowchart Interconnections



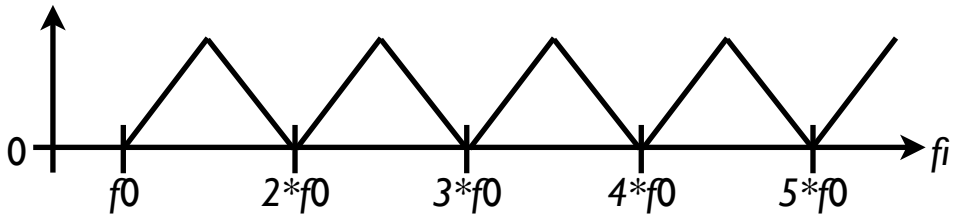
The '***Contrast***' parameter associated to ***mirpitch*** can be specified, and is set by default to .3

*mir**inharmonicity***

Partials non-multiple of fundamentals

mirinharmonicity(x) estimates the inharmonicity, i.e., the amount of partials that are not multiples of the fundamental frequency, as a value between 0 and 1. More precisely, the inharmonicity considered here takes into account the amount of energy outside the ideal harmonic series.

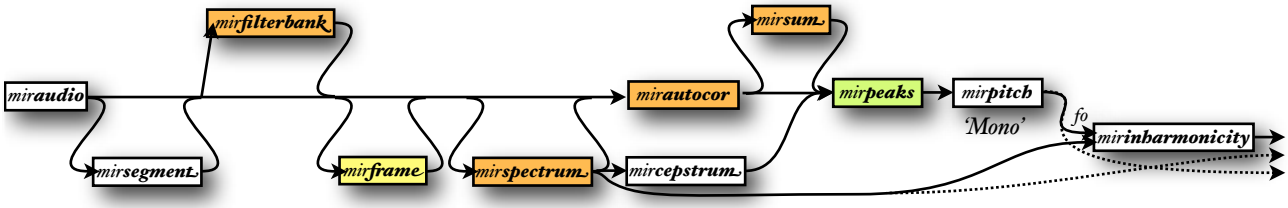
For that purpose, we use a simple function estimating the inharmonicity of each frequency given the fundamental frequency *f*0:



WARNING:

This simple model presupposes that there is only one fundamental frequency.

Flowchart Interconnections



mirinharmonicity accepts as main input either:

- ***mirspectrum*** objects,
- ***miraudio*** objects (same as for *mirspectrum*),
- **file name** or the '***Folder***' keyword.

mirinharmonicity can return several outputs:

1. the inharmonicity rate itself,
2. the ***mirspectrum*** data, and

3. the fundamental frequency '***fo***'.

Option

*mirinharmonicity(..., '**fo**', f)* bases the computation of the inharmonicity on the fundamental frequency indicated by *f*. The frequency data can be either a number, or a *mirscalar* object (for instance, the output of a *mirpitch* computation).

By default, the fundamental frequency is computed using the command:

f = mirpitch(..., 'Mono')

mirchromagram

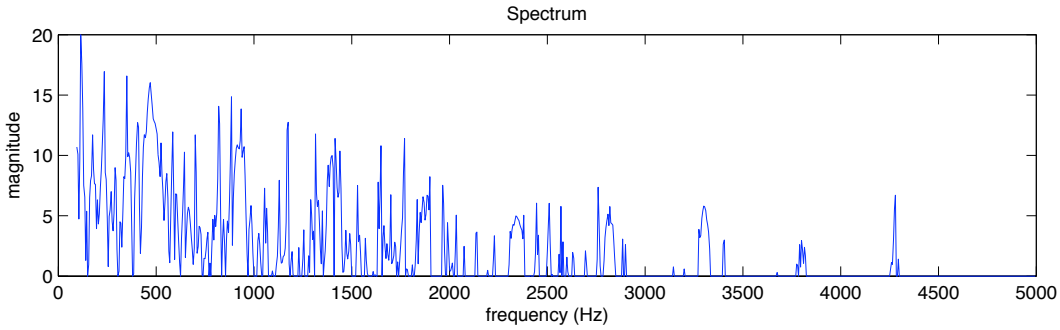
Energy distribution along pitches

The chromagram, also called Harmonic Pitch Class Profile, shows the distribution of energy along the pitches or pitch classes.

- First the spectrum is computed in the logarithmic scale, with selection of, by default, the 20 highest dB, and restriction to a certain frequency range that covers an integer number of octaves, and normalization of the audio waveform before computation of the FFT.

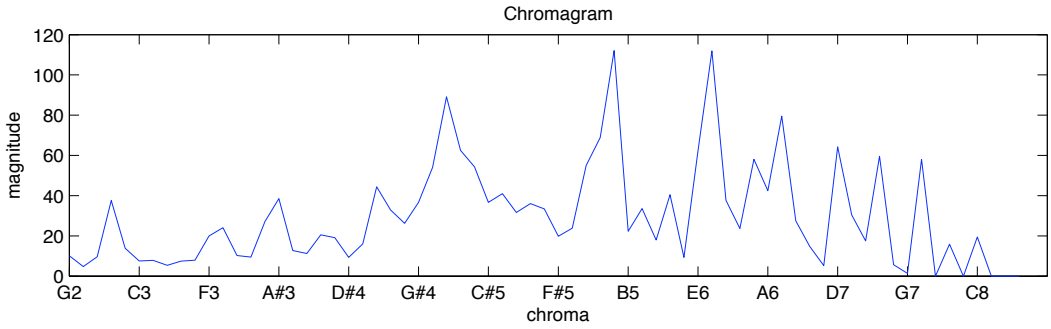
s = mirspectrum(..., 'dB', 20, 'Min', fmin, 'Max', fmax, 'NormalInput', 'MinRes', r, 'OctaveRatio', .85)

  - The minimal frequency *fmin* is specified by the '*Min*' option (cf. below).
  - The maximal frequency *fmax* is at least equal to the '*Max*' option, but the frequency range is extended if necessary in order to obtain an integer number of octaves.
  - The minimal frequency resolution of the spectrum is chosen such that even lowest chromas will be segregated from the spectrum. It is based on the number of chromas per octave *r* ('*Res*' option , cf. below). Lowest frequencies are ignored if they do not meet this resolution constraint, and a warning message is sent by *mirspectrum* (cf. '*OctaveRatio*' keyword).



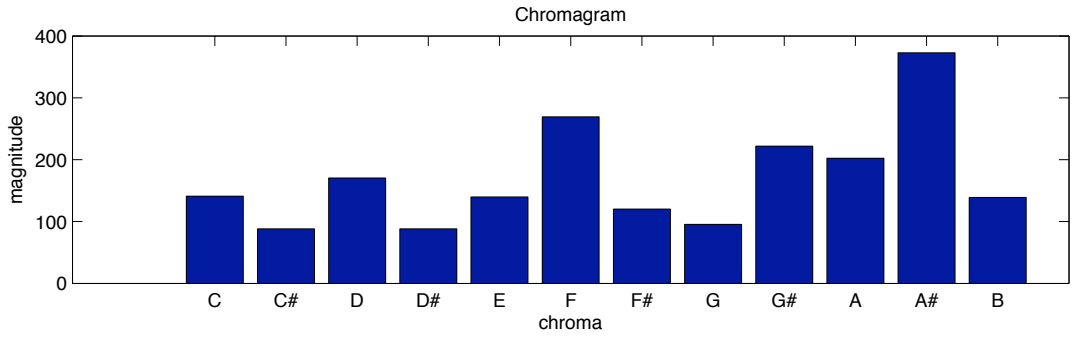
- The chromagram is a redistribution of the spectrum energy along the different pitches (i.e., "chromas"):

c = mirchromagram(s, 'Wrap', 'no')

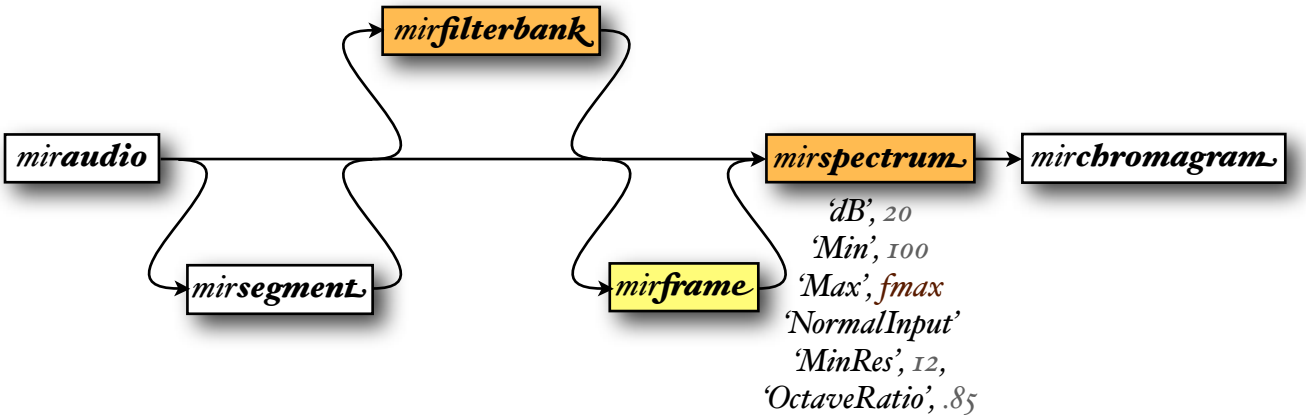


• If the '***Wrap***' option is selected, the representation is wrapped along the 12 pitch classes:

c = mirchromagram(c, 'Wrap', 'yes')



FLOWCHART INTERCONNECTIONS



The '***Min***' and '***Max***' range used in ***mirspectrum*** can be tuned directly in *mirchromagram*, as well as the '***dB***' threshold (that can be written '***Threshold***' as well). These parameters are set by default to *Min* = 100 Hz, *Max* = 5000 Hz (Gómez, 2006) and *Threshold* = 20 dB. However, it seems that the spectrum should span as closely as possible an integer number of octaves, in order to avoid emphasizing particular pitches covered more often than others. Hence the higher limit of the frequency range of the spectrum computation is increased accordingly. Hence for the default parameters value (*Min* = 100 Hz, *Max* = 5000 Hz), the actual maximum frequency

fmax is set to 6400 Hz. The '***MinRes***' value corresponds to the '*Res*' chromagram resolution parameter, as explained in the option section below.

mirchromagram~ accepts either:

- ***mirspectrum~*** objects,
- ***miraudio*** objects , where the audio waveform can be segmented (using ***mirsegment~***), decomposed into channels (using ***mirfilterbank***), and/or decomposed into frames (using ***mirframe~*** or the *'**Frame**'* option, with by default a frame length of 200 ms and a hop factor of .05),
- **file name** or the *'**Folder**'* keyword.

Options

- *c = mirchromagram(..., '**Tuning**', t)* specifies the frequency (in Hz.) associated to chroma C. Default value, t = 261.6256 Hz.
- *c = mirchromagram(..., '**Triangle**')* weights the contribution of each frequency with respect to the distance with the actual frequency of the corresponding chroma.
- *c = mirchromagram(..., '**Weight~**', o)* specifies the relative radius of the weighting window, with respect to the distance between frequencies of successive chromas.
  - *o* = 1: each window begins at the centers of the previous one.
  - *o* = .5: each window begins at the end of the previous one. (default value)
- *c = mirchromagram(..., '**Res**', r)* indicates the resolution of the chromagram in number of bins per octave. Default value, *r* = 12.

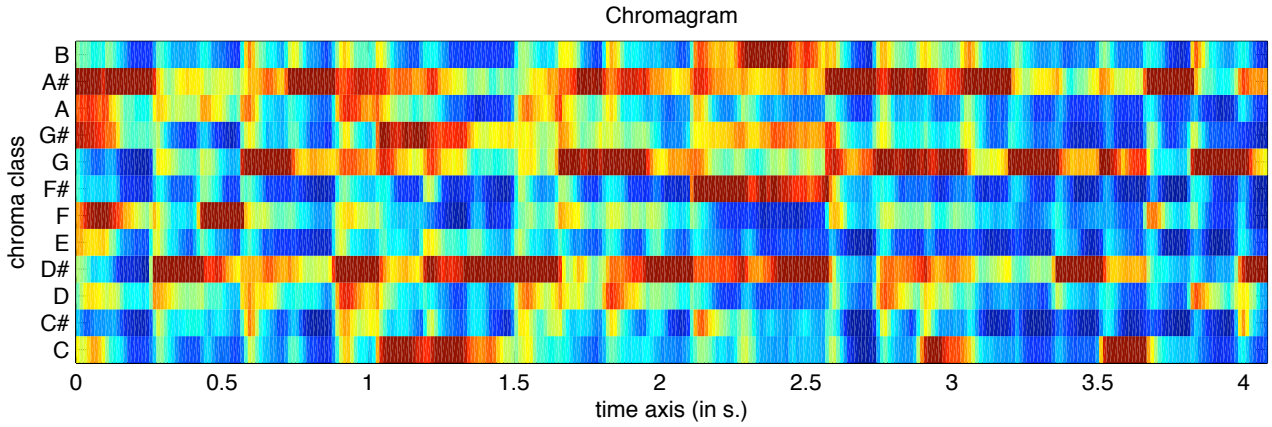
Post-processing operations

- *c = mirchromagram(..., '**Wrap**', w)* specifies whether the chromagram is wrapped or not.
  - *w* = *'yes'*: groups all the pitches belonging to same pitch classes (default value)
  - *w* = *'no'*: pitches are considered as absolute values.
- *c = mirchromagram(..., '**Center**')* centers the result.
- *c = mirchromagram(..., '**Normal**', n)* normalizes the result, using the *n~*-norm. The default value is *n~* = *Inf*, corresponding to a normalization by the maximum value. *n~* = 0 toggles off the normalization. Alternative keyword: '***Norm~***'.
- *c = mirchromagram(..., '**Pitch**', p)* specifies how to label chromas in the figures.

- *p* = *'yes'*: chromas are labeled using pitch names (default)
- *p* = *'no'*: chromas are labeled using MIDI pitch numbers.

EXAMPLE

mirchromagram('ragtime', 'Frame')



ACCESSIBLE OUTPUT

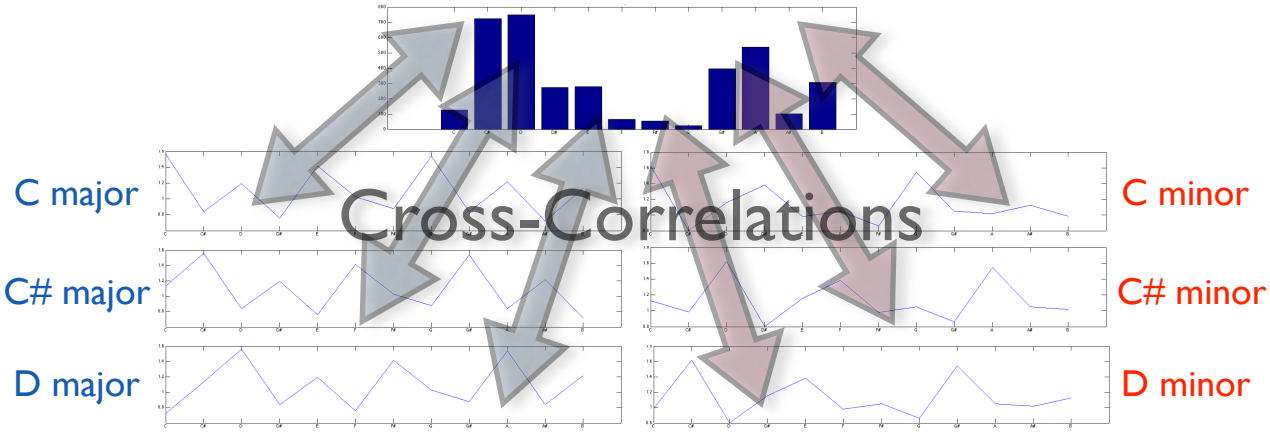
cf. §5.2 for an explanation of the use of the *get* method. Specific fields:

- '***Magnitude***': same as '*Data*',
- '***Chroma***': the chroma related to each magnitude (same as '*Pos*'),
- '***ChromaClass***': the chroma class ('A', 'A#', 'B', etc.) related to each chroma,
- '***ChromaFreq***': the central frequency of each chroma, in the unwrapped representation,
- '***Register***': the octave position,
- '***PitchLabel***': whether the chroma are represented as simple numeric positions (0), or as couples (ChromaClass, Register) (1).
- '***Wrap***': whether the chromagram is represented along all possible chromas (0), or along the 12 chroma-classes only (1).

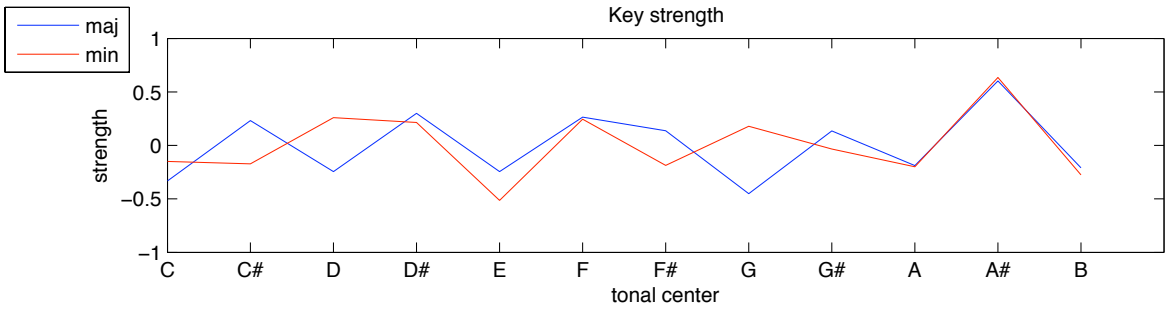
mirkeystrength

Probability of key candidates

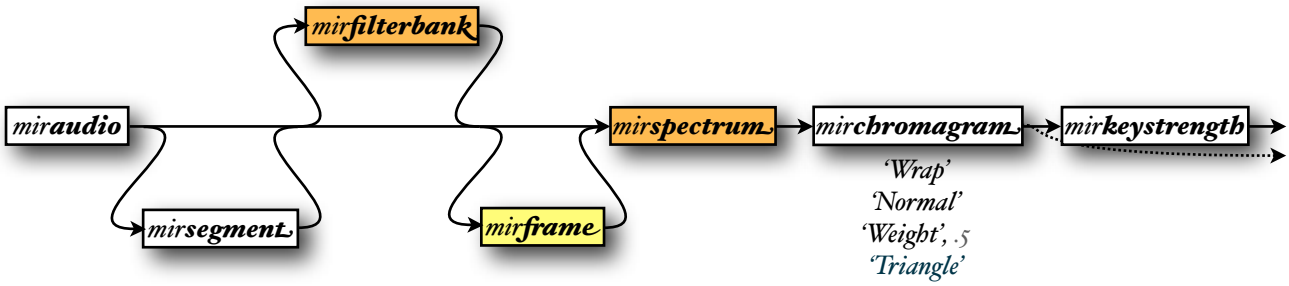
mirkeystrength computes the key strength, a score between -1 and +1 associated with each possible key candidate, through a cross-correlation of the chromagram returned by *mirchromagram,* wrapped and normalized (using the '*Normal*' option), with similar profiles representing all the possible tonality candidates (Krumhansl, 1990; Gomez, 2006).



The resulting graph indicate the cross-correlation score for each different tonality candidate.



Flowchart Interconnections



For the moment, only the '***Weight***' and '***Triangle***' options used in ***mirchromagram*** can be tuned directly in *mirkeystrength*.

mirkeystrength accepts either:

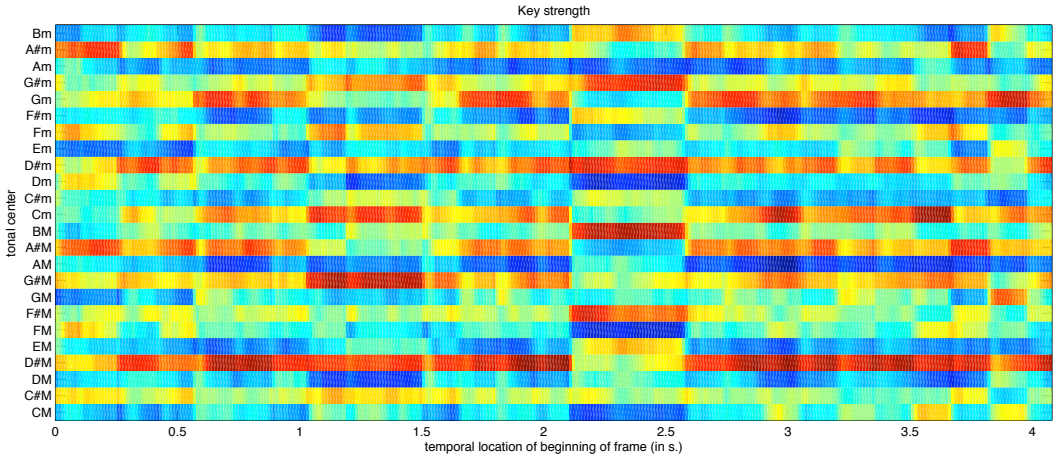
- ***mirchromagram~*** objects,
- ***mirspectrum~*** objects,
- ***miraudio*** objects (same as for *mirchromagram~*),
- **file name** or the '***Folder***' keyword.

mirkeystrength can return several outputs:

1. the key strength itself, and
2. the ***mirchromagram~*** data.

Example

mirkeystrength('ragtime', 'Frame')



Accessible Output

cf. §5.2 for an explanation of the use of the *get~* method. Specific fields:

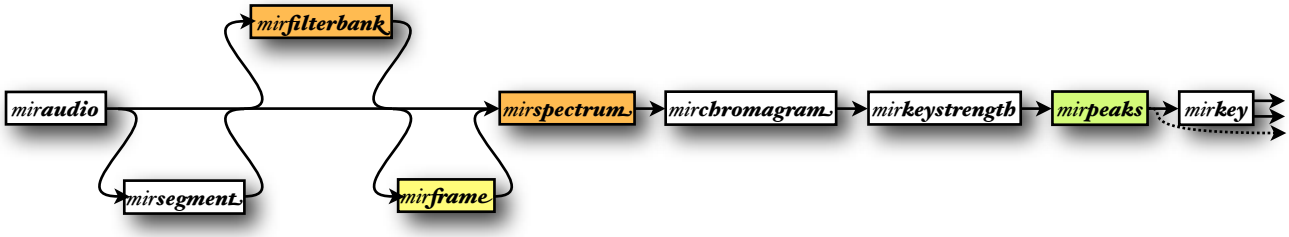
- '***Strength***': the key strength value for each key and each temporal position (same as '*Data*'). The resulting matrix is distributed along two layers along the fourth dimension: a first layer for major keys, and a second layer for minor keys.
- '***Tonic***': the different key centres (same as '*Pos*').

mirkey

Description

Gives a broad estimation of tonal center positions and their respective clarity.

Flowchart Interconnections



It consists simply of a peak picking in the *mirkeystrength* curve(s). Two options of ***mirpeaks*** are accessible from *mirkey*:

- '***Total***', set to 1
- '***Contrast***', set to .1

The '***Weight***' and '***Triangle***' options used in ***mirchromagram*** can be changed directly in *mirkeystrength*.

mirkey accepts either:

- ***mirkeystrength*** objects, where peaks have been already extracted or not,
- ***mirchromagram*** objects,
- ***mirspectrum*** objects,
- ***miraudio*** objects , where the audio waveform can be segmented (using ***mirsegment***), decomposed into channels (using ***mirfilterbank***), and/or decomposed into frames (using ***mirframe*** or the '***Frame***' option, with by default a frame length of 1 s and half overlapping),
- **file name** or the '***Folder***' keyword.

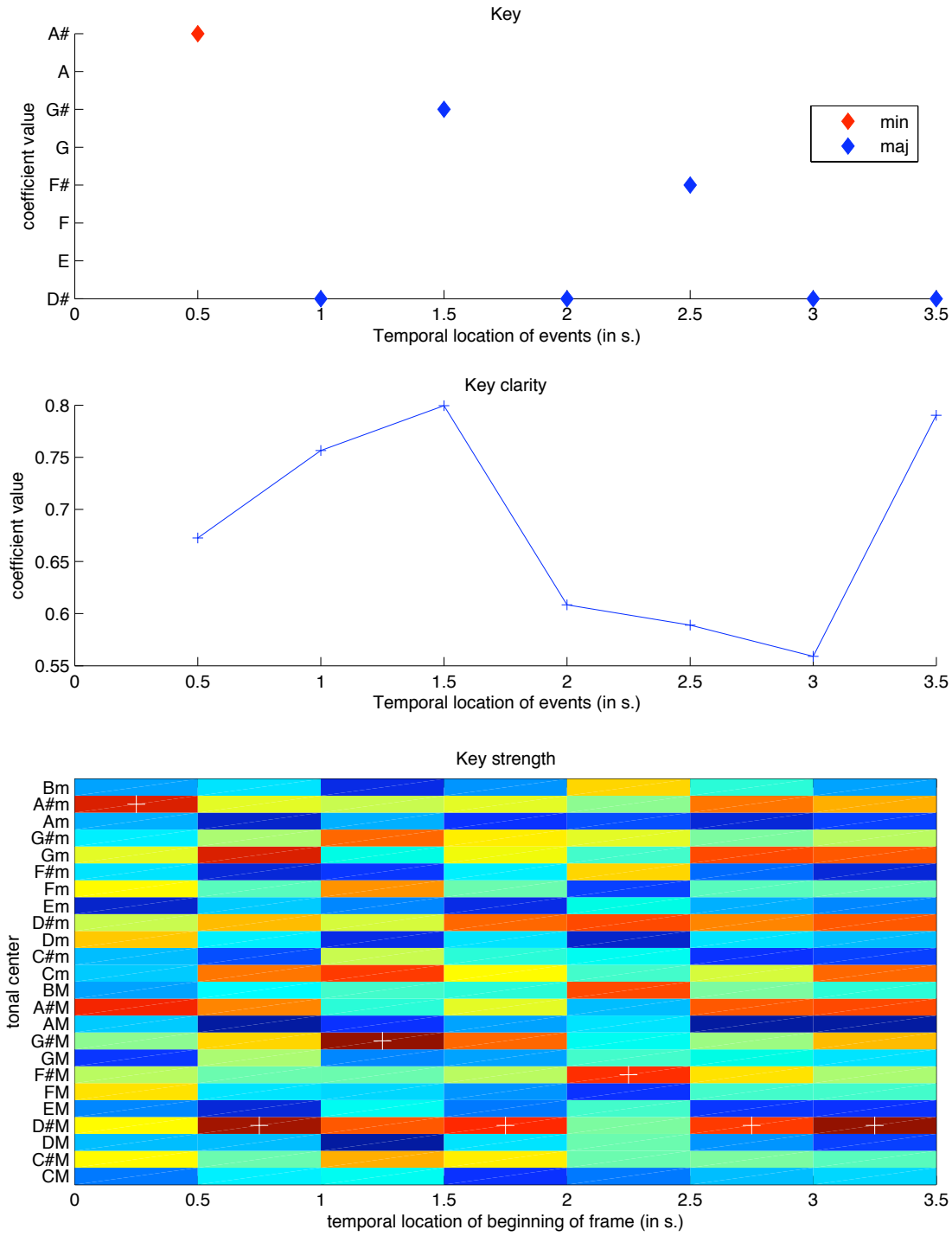
mirkey can return several outputs:

1. the best key(s), i.e., the peak abscissa(e);
2. the *key clarity*: i.e., the key strength associated to the best key(s), i.e., the peak ordinate(s);

3. the ***mirkeystrength*** data including the picked peaks (***mirpeaks***).

EXAMPLE

[k c s] = mirkey('ragtime', 'Frame')

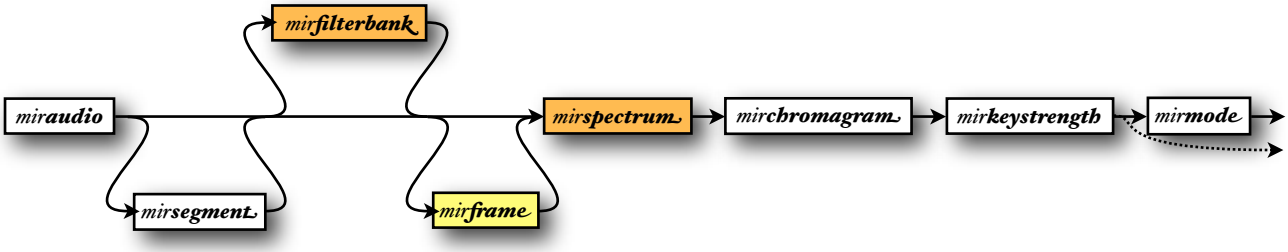


mirmode

Description

Estimates the modality, i.e. major vs. minor, returned as a numerical value between -1 and +1: the closer it is to +1, the more major the given excerpt is predicted to be, the closer the value is to -1, the more minor the excerpt might be.

Flowchart Interconnections



mirmode accepts either:

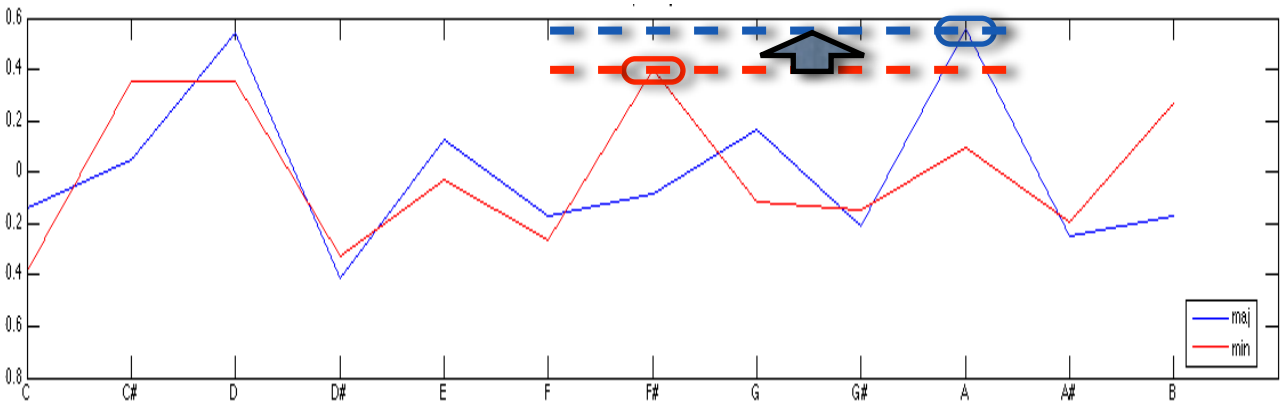
- ***mirkeystrength*** objects, where peaks have been already extracted or not,
- ***mirchromagram*** objects,
- ***mirspectrum*** objects,
- ***miraudio*** objects (same as for *mirkey*) or
- **file name** or the '***Folder***' keyword.

mirmode can return several outputs:

1. modality itself, and
2. the ***mirkeystrength*** result used for the computation of modality.

Strategies

- *mirkeystrength*(..., '***Best***') computes the key strength difference between the best major key (highest key strength value) and the best minor key (lowest key strength value). (default choice)



- *mirkeystrength(..., '**Sum**')* sums up the key strength differences between all the major keys and their relative minor keys.

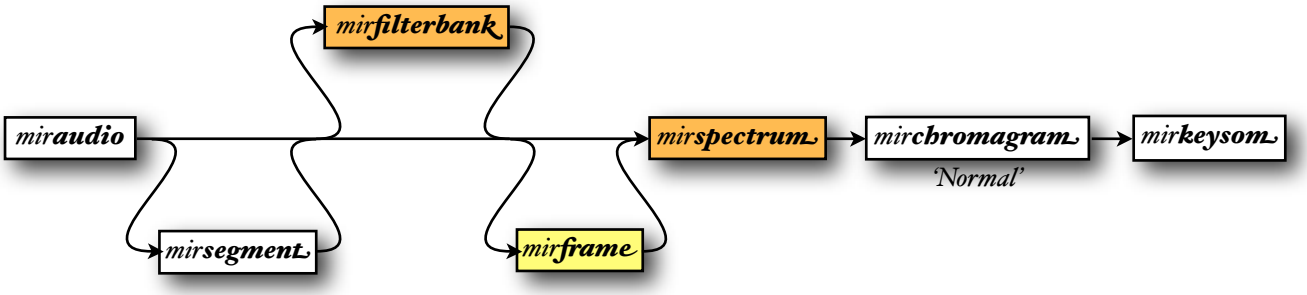
mirkeysom

Description

Projects the chromagram (normalized using the '*Normal*' option) into a self-organizing map trained with the Krumhansl-Kessler profiles (modified for chromagrams) (Toiviainen and Krumhansl, 2003; Krumhansl, 1990).

The result is displayed as a pseudo-color map, where colors correspond to Pearson correlation values. In case of frame decomposition, the projection maps are shown one after the other in an animated figure.

Flowchart Interconnections

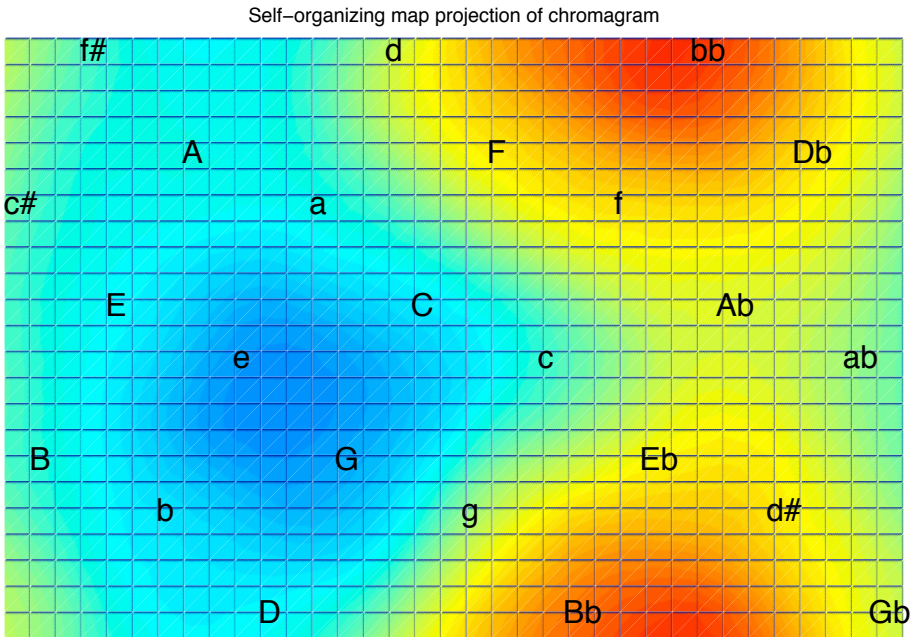


mirkeysom accepts either:

- ***mirchromagram*** objects,
- ***mirspectrum*** objects,
- ***miraudio*** objects , where the audio waveform can be segmented (using ***mirsegment***), decomposed into channels (using ***mirfilterbank***), and/or decomposed into frames (using ***mirframe*** or the '***Frame***' option, with by default a frame length of 1 s and half overlapping),
- **file name** or the '***Folder***' keyword.

Example

mirkeysom('ragtime')



Accessible Output

cf. §5.2 for an explanation of the use of the *get.* method. Specific fields:

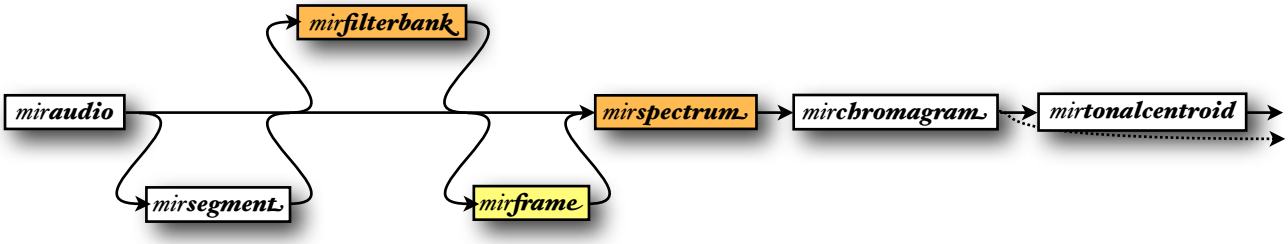
• '***Weight.***': the projection value associated to each map location (same as '*Data*').

mirtonalcentroid

Description

Calculates the 6-dimensional tonal centroid vector from the chromagram. It corresponds to a projection of the chords along circles of fifths, of minor thirds, and of major thirds (Harte and Sandler, 2006).

Flowchart Interconnections



mirtonalcentroid accepts either:

- ***mirchromagram*** objects,
- ***mirspectrum*** objects,
- ***miraudio*** objects , where the audio waveform can be segmented (using ***mirsegment***), decomposed into channels (using ***mirfilterbank***), and/or decomposed into frames (using ***mirframe*** or the '***Frame***' option, with by default a frame length of .743 s and a hop factor of .1),
- **file name** or the '***Folder***' keyword.

mirtonalcentroid can return several outputs:

1. the tonal centroid itself, and
2. the ***mirchromagram*** data.

Accessible Output

cf. §5.2 for an explanation of the use of the *get* method. Specific fields:

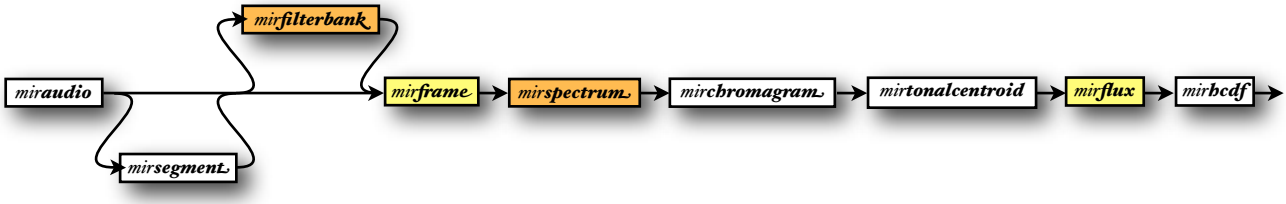
- '***Dimensions***': the index of the 6 dimensions (same as '*Pos*'),
- '***Positions***': the position of each data within the 6-dimensional space (same as '*Data*').

mirhcdf

Description

The Harmonic Change Detection Function (HCDF) is the flux of the tonal centroid (Harte and Sandler, 2006).

Flowchart Interconnections



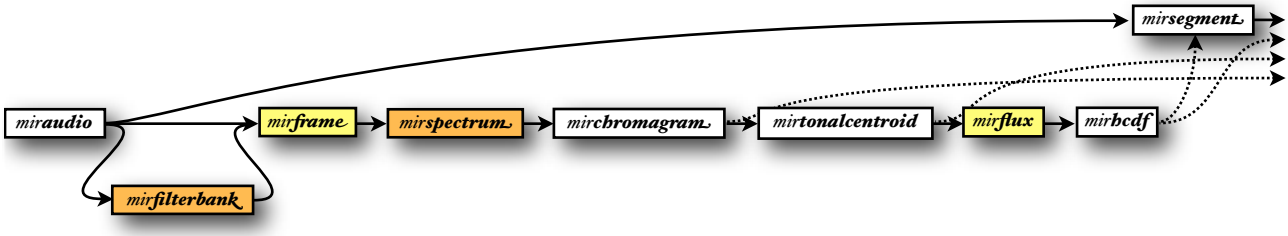
mirhcdf accepts either:

- ***mirtonalcentroid*** frame-decomposed objects,
- ***mirchromagram*** frame-decomposed objects,
- ***mirspectrum*** frame-decomposed objects,
- ***miraudio*** objects , where the audio waveform can be segmented (using ***mirsegment***), decomposed into channels (using ***mirfilterbank***). If not decomposed yet, it is decomposed into frames (using the '***Frame***' option, with by default a frame length of .743 s and a hop factor of .1),
- **file name** or the '***Folder***' keyword.

mirsegment(..., 'HCDF')

Peak detection applied to the HCDF returns the temporal position of tonal discontinuities that can be used for the actual segmentation of the audio sequence.

Flowchart Interconnections



mirsegment accepts uniquely as main input a ***miraudio*** objects not frame-decomposed, not channel decomposed, and not already segmented. Alternatively, **file name** or the '***Folder***' keyword can be used as well.

mirsegment(..., 'HCDF') can return several outputs:

1. the segmented audio waveform itself,
2. the HCDF (***mirhcdf***) after peak picking (***mirpeaks***),
3. the tonal centroid (***mirtonalcentroid***), and
4. the chromagram (***mirchromagram***).

4. HIGH-LEVEL FEATURES

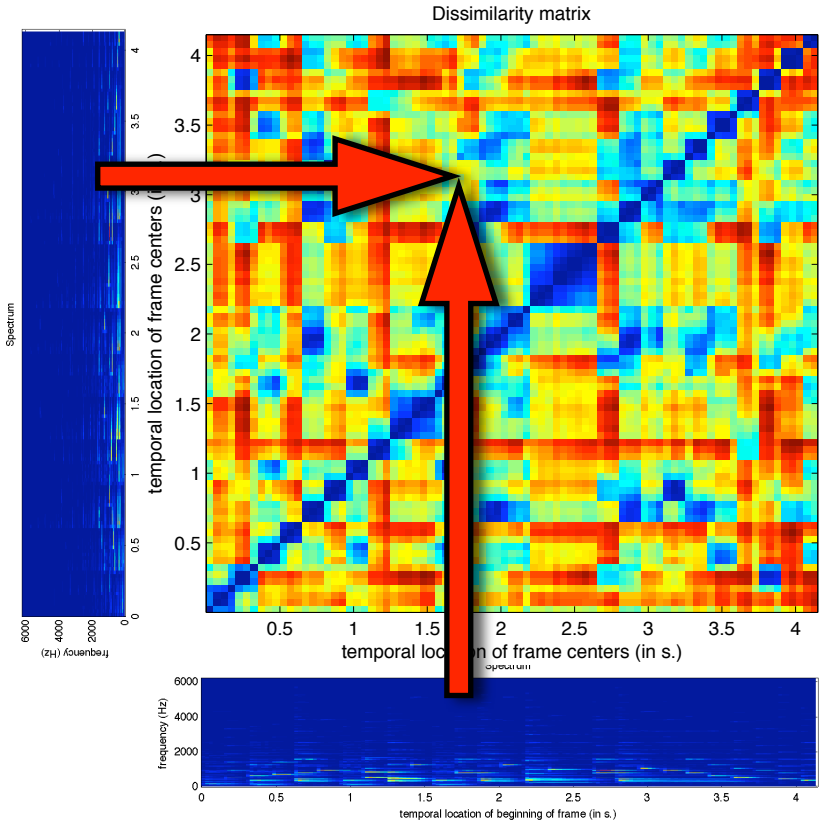
4.1. Structure and form

More elaborate tools have also been implemented that can carry out higher-level analyses and transformations. In particular, audio files can be automatically segmented into a series of homogeneous sections, through the estimation of temporal discontinuities along diverse alternative features such as timbre in particular (Foote and Cooper, 2003).

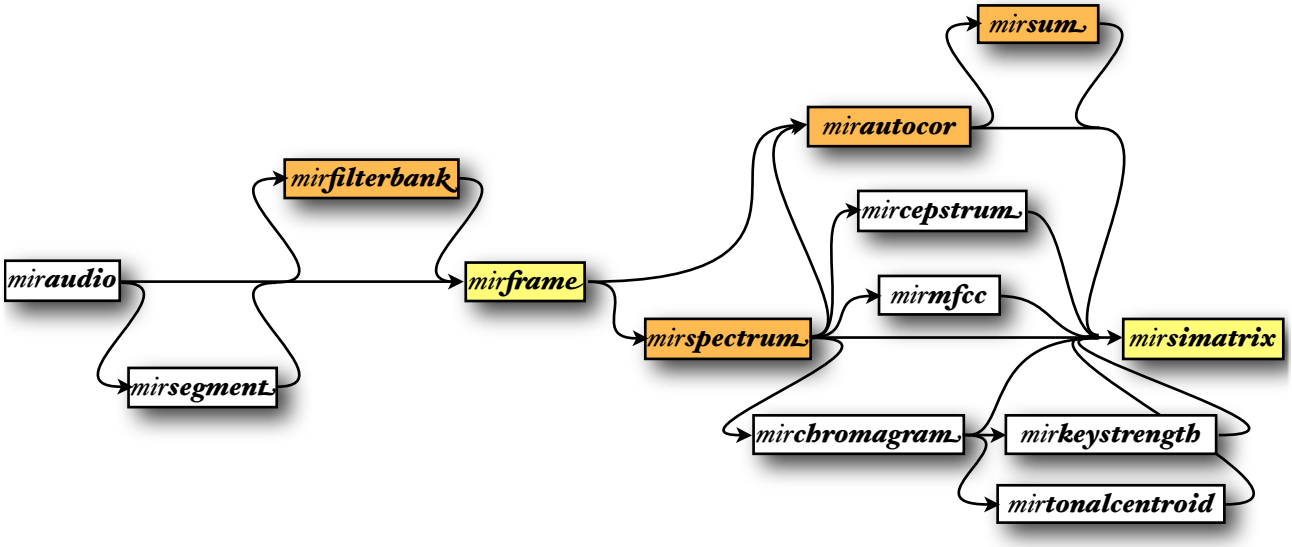
mirsimatrix

DESCRIPTION

A similarity matrix shows the similarity between all all possible pairs of frames from the input data.


Dissimilarity matrix
Spectrum
frequency (Hz)
temporal location of frame centers (in s.)
temporal location of beginning of frame (in s.)

Flowchart Interconnections



mirsimatrix usually accepts either:

- ***mirspectrum*** frame-decomposed objects.
- ***miraudio*** objects: in this case, the (dis)similarity matrix will be based on the spectrogram (***mirspectrum***). The audio waveform is decomposed into frames if it was not decomposed yet, and the default frame parameters – frame length of 50 ms and no overlapping – can be changed using the '***Frame***' option.

If the audio waveform is segmented (with ***mirsegment***), the similarities are not computed between frames but on the contrary between segments, using Kullback-Leibler distance (Foote and Cooper, 2003). *(Removed since version 1.2, due to difficulties that have not been solved since.)*

- **file name** or the '***Folder***' keyword: same behavior than for *miraudio* objects;
- ***mirautocor*** frame-decomposed objects;
- ***mircepstrum*** frame-decomposed objects;
- ***mirmfcc*** frame-decomposed objects;
- ***mirchromagram*** frame-decomposed objects;
- ***mirkeystrength*** frame-decomposed objects;
- a *Matlab* square matrix: in this case, a *mirsimatrix* object is created based on the matrix given as input, with the following options:
  - The frame rate used for the matrix can be specified as second input of *mirsimatrix*. By default, it is set to 20 Hz.

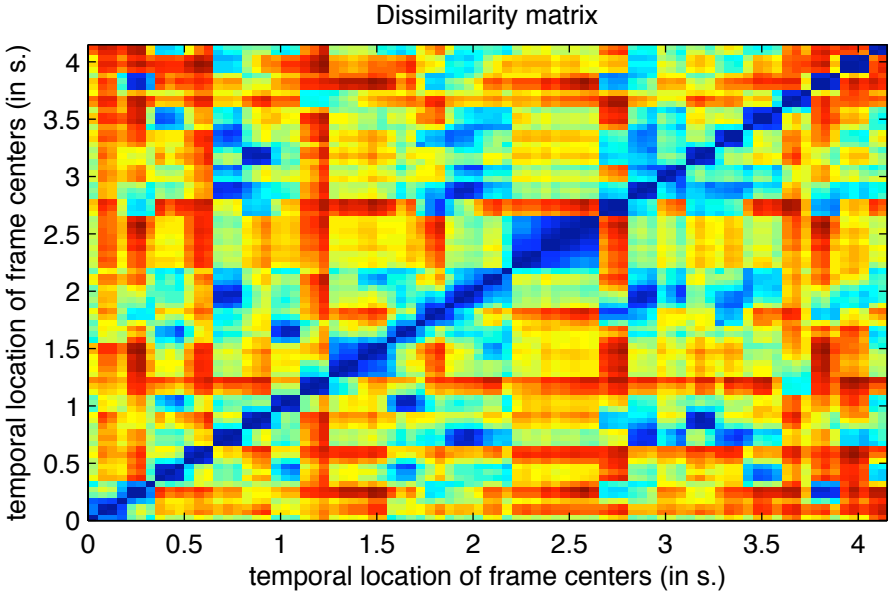
- The input is supposed to be a *similarity* matrix already. If the input is in fact a *dissimilarity* matrix, the keyword *'Dissimilarity'* should be added as third argument of *mirsimatrix*.

Options

- *mirsimatrix(..., '**Distance**', f)* specifies the name of a distance function, from those proposed in the *Statistics Toolbox* (help *pdist*). default value: *f = 'cosine'*

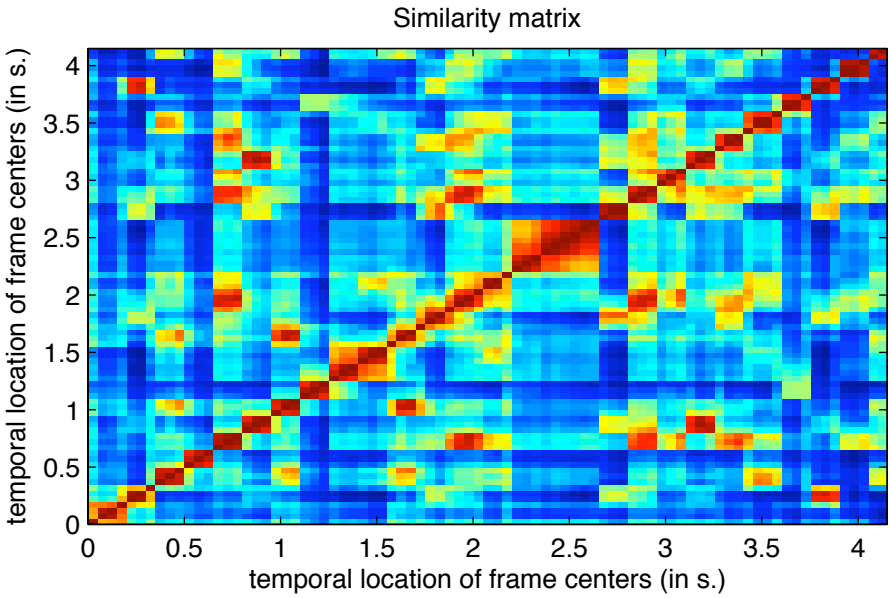
- *mirsimatrix(..., '**Width**', w)* specifies the size of the diagonal bandwidth, in samples, outside which the dissimilarity will not be computed. If *w* is even, the actual width is *w*-1 samples. If *w = inf* (default value), all the matrix will be computed.

- *mirsimatrix(..., '**Dissimilarity**')* return the *dissimilarity matrix*, which is the intermediary result before the computation of the similarity matrix. It shows the distance between each possible frame.

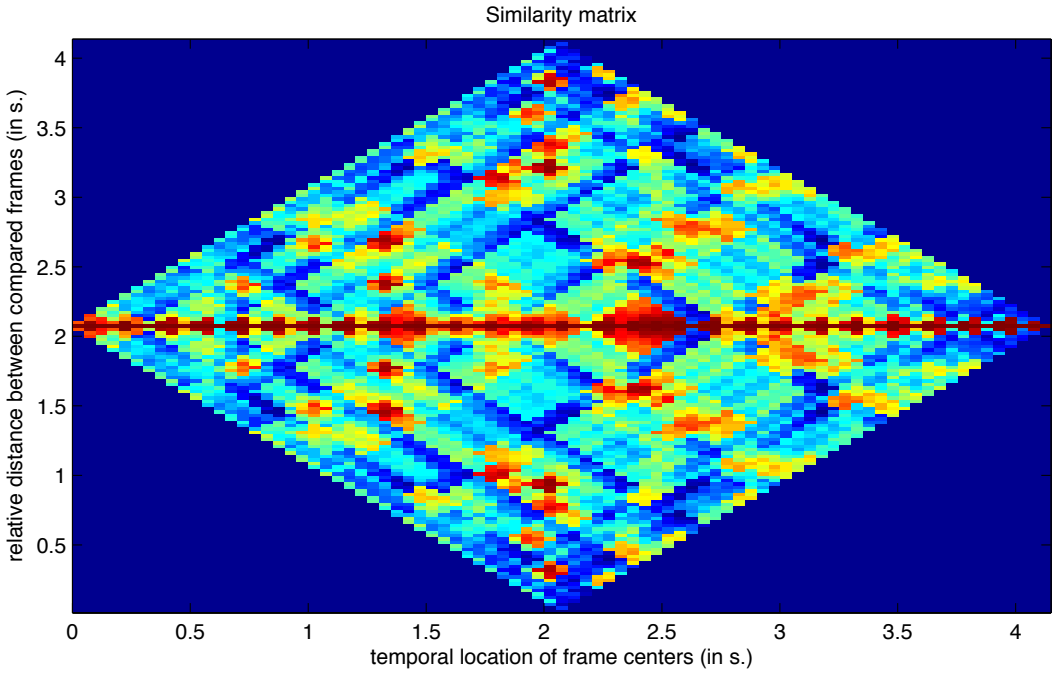


- *mirsimatrix(..., '**Similarity**', f)* indicates the function *f* specifying the way the distance values in the dissimilarity matrix are transformed into the values of the *similarity matrix*. default value: *f = 'exponential'* corresponding to

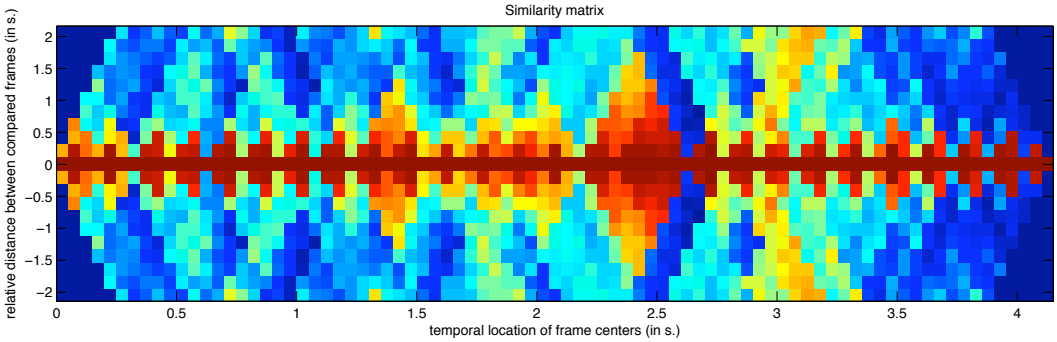
$$f(x) = exp(-x)$$



- *mirsimatrix(..., '**Horizontal**')* rotates the matrix 45° in order to make the first diagonal horizontal:

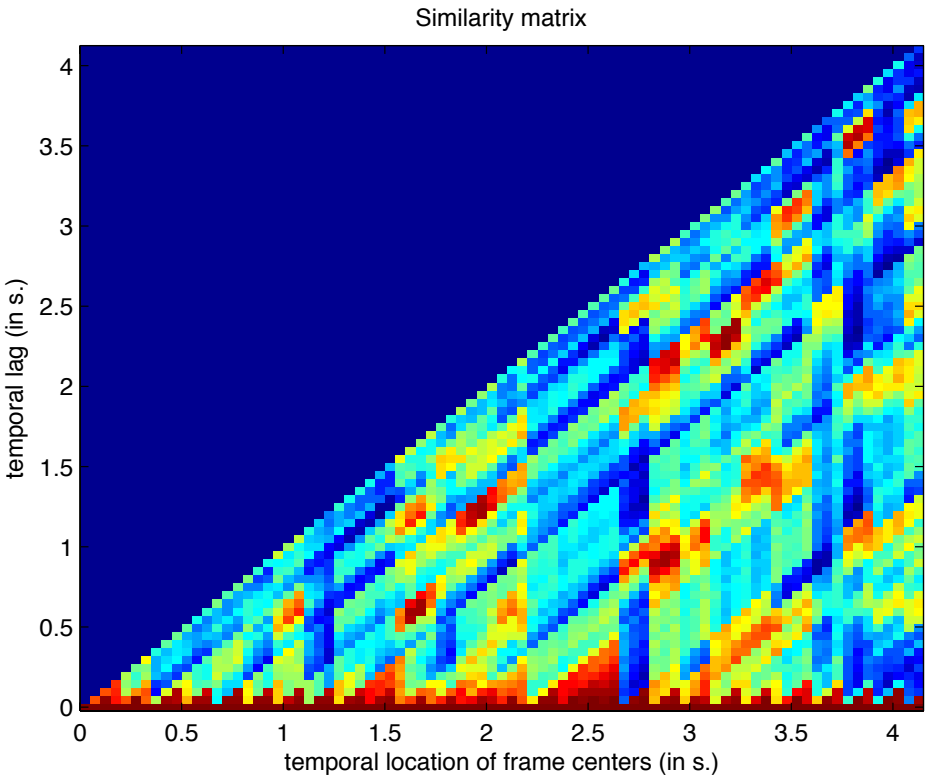


- *mirsimatrix(..., '**Horizontal**', '**Width**', w)* enables to restrict to a diagonal bandwidth only.

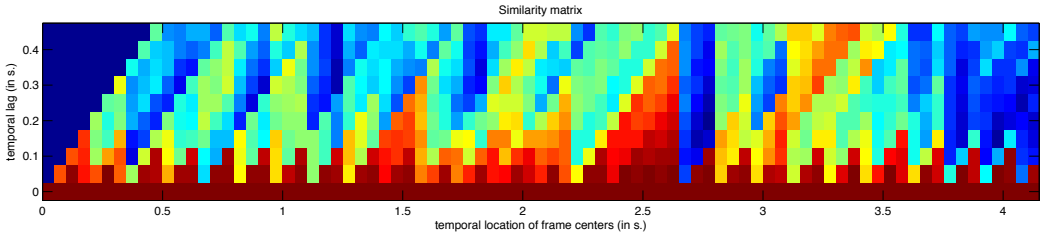


mirsimatrix(..., 'Width', 20, 'Horizontal')

- *mirsimatrix(..., '**TimeLag**')* computes the time-lag matrix out of the (non-rotated) initial time-time matrix:



- *mirsimatrix(..., '**TimeLag**', '**Width**', w)* enables to restrict to a diagonal bandwidth only:



mirsimatrix(..., 'Width', 20, 'TimeLag')

ACCESSIBLE OUTPUT

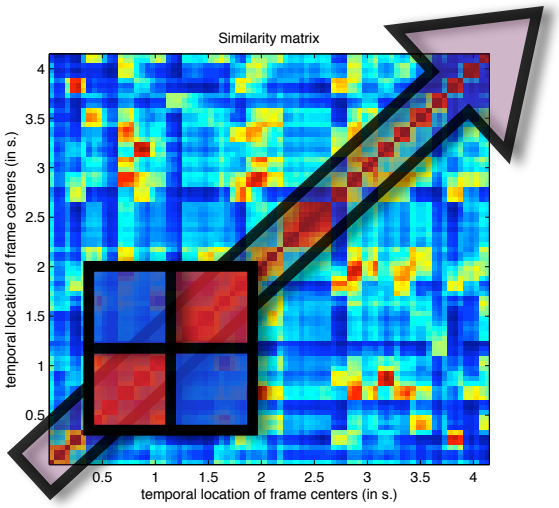
cf. §5.2 for an explanation of the use of the *get* method. Specific field:

• '***DiagWidth***': the chosen value for the '*Width*' parametre.

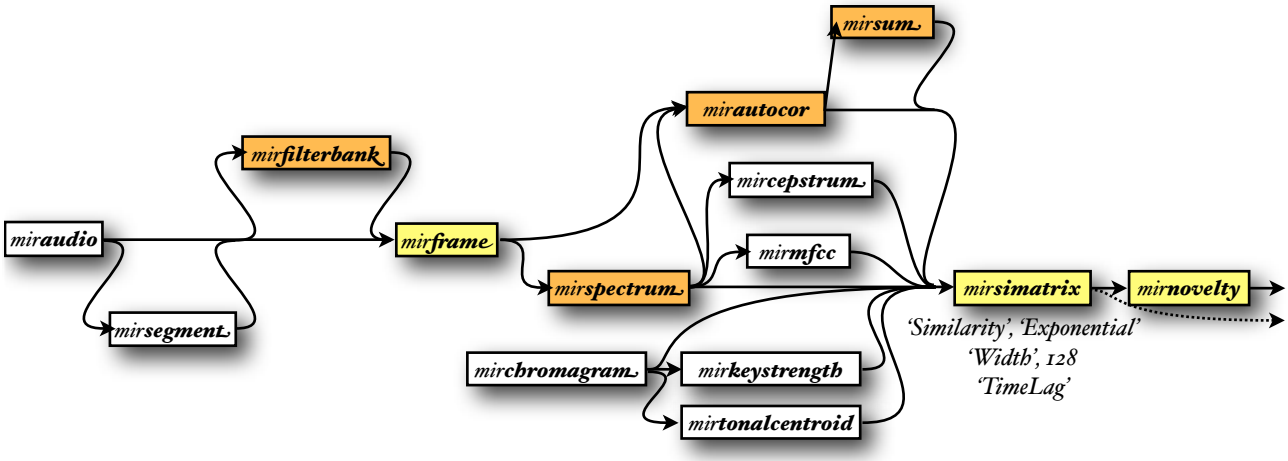
mirnovelty

NOVELTY CURVE

Convolution along the main diagonal of the similarity matrix using a Gaussian checkerboard kernel yields a novelty curve that indicates the temporal locations of significant textural changes.



FLOWCHART INTERCONNECTIONS



Some parameters related to ***mirsimatrix*** are accessible in *mirnovelty*:

- '***Distance***',
- '***Similarity***' (set here to the default value '*exponential*'),
- '***Width***' (can also be called '***KernelSize***'), with the default value here 64 samples,
- '***Horizontal***', instead of the default '***TimeLag***'.

mirnovelty usually accepts either:

- ***mirsimatrix*** objects,
- ***mirspectrum*** frame-decomposed objects,
- ***miraudio*** objects (same as for *mirsimatrix*).
- **file name** or the '***Folder***' keyword: same behavior than for *miraudio* objects;
- ***mirautocor*** frame-decomposed objects;
- ***mircepstrum*** frame-decomposed objects;
- ***mirmfcc*** frame-decomposed objects;
- ***mirchromagram*** frame-decomposed objects;
- ***mirkeystrength*** frame-decomposed objects.

mirnovelty can return several outputs:

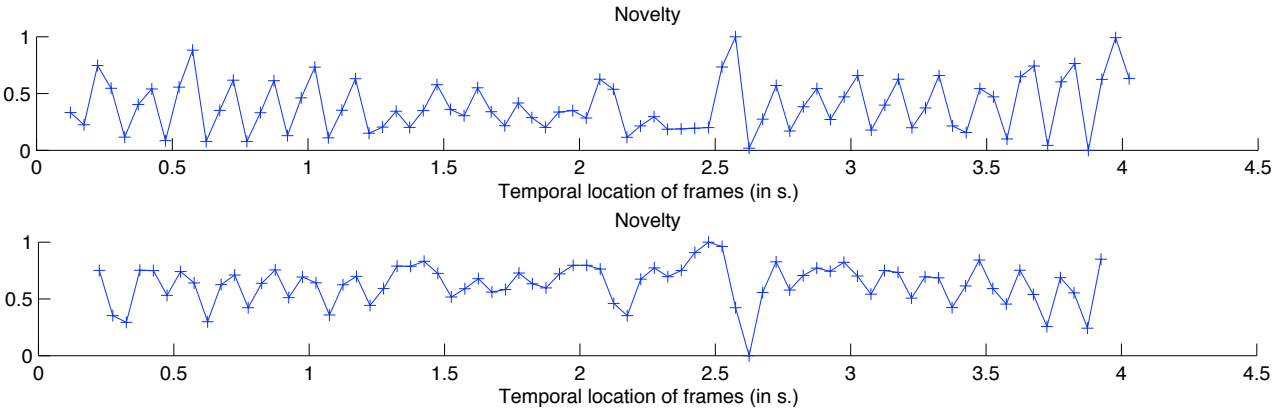
1. the novelty curve itself, and
2. the similarity matrix (***mirsimatrix***), horizontal.

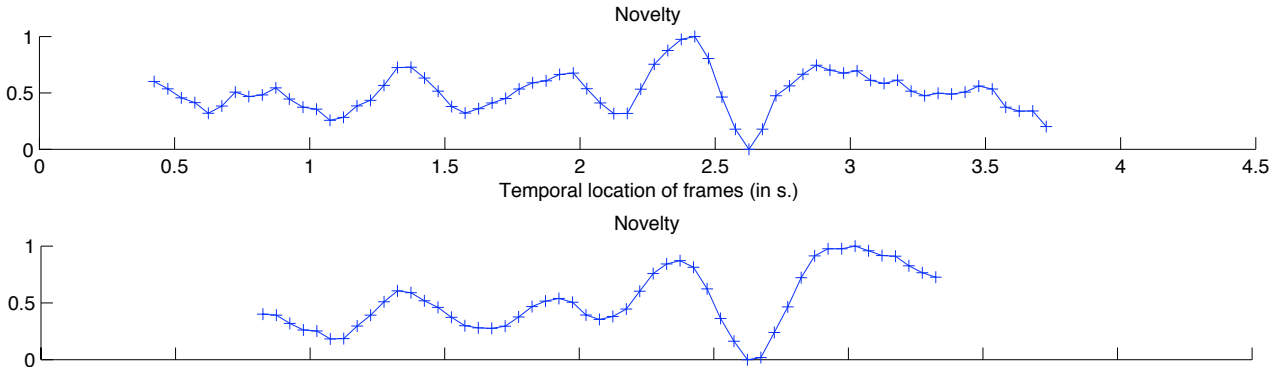
POST-PROCESSING OPERATION

- *mirnovelty(..., '**Normal**', n)* toggles on/off the normalization of the novelty curve between the values 0 and 1. Toggled on by default.

EXAMPLE

Novelty curve computed using increasing kernel size '*KernelSize*':



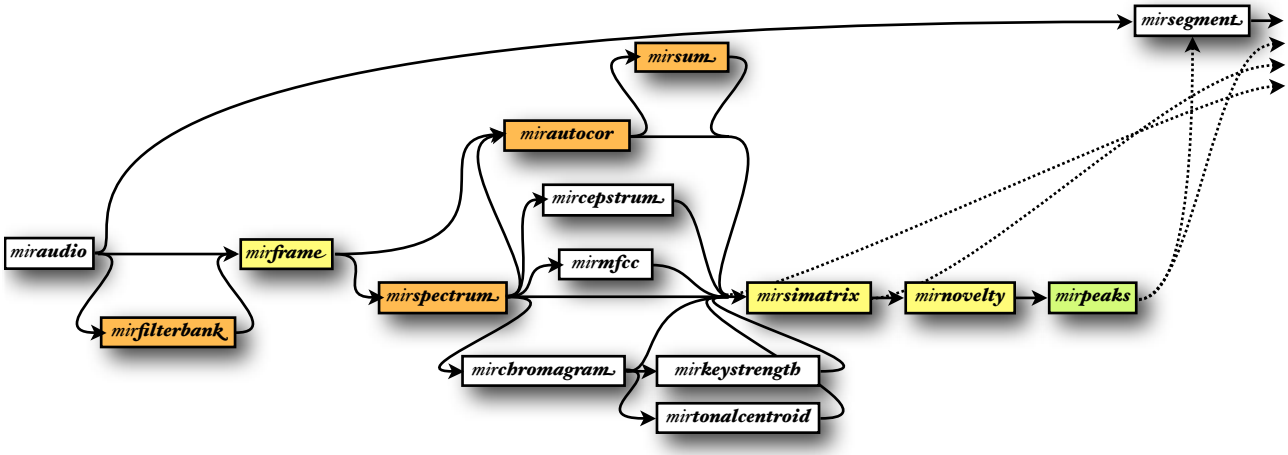

Novelty
1
0.5
0
0
0.5
1
1.5
2
2.5
3
3.5
4
4.5
Temporal location of frames (in s.)
Novelty
1
0.5
0

mirsegment(..., 'Novelty')

Peak detection applied to the novelty curve returns the temporal position of feature discontinuities that can be used for the actual segmentation of the audio sequence.

The '*Novelty*' keyword is actually not necessary, as this strategy is chosen by default in *mirsegment*.

Flowchart Interconnections



Some parameters related to ***mirnovelty*** are accessible in *mirsegment*: '***Distance***', '***Measure***' and '***KernelSize***'. Some parameters related to ***mirpeaks*** are accessible in *mirsegment*: '***Total***' (set by default to *Inf*) and '***Contrast***' (set to .1).

The choice of the feature used for the similarity matrix computation can be specified:

- *mirsegment(..., '**Spectrum**')* will compute a ***mirspectrum***, where some parameters can be specified: '*Min*', '*Max*', '*Normal*', '*Window*', '*Bark*', '*Mel*'. The default frame length is 50 ms and no overlapping.
- *mirsegment(..., '**MFCC**')* will compute a ***mirmfcc***, where the '*Rank*' parameter can be specified. Same default frame parameters than for '*Spectrum*'.
- *mirsegment(..., '**KeyStrength**')* will compute a ***mirkeystrength***. The default frame length is 500 ms and a hop factor of .2
- *mirsegment(..., '**Pitch**')* ou *mirsegment(..., '**AucotorPitch**')* will compute a ***mirpitch*** operation and use its second output, i.e., the ***mirautocor***, for the computation of the similarity matrix. The default frame length is 50 ms and a hop factor of .01
- If no feature is specified, the default feature used in *mirsimatrix* will be chosen, i.e., the spectrum (***mirspectrum***).

The default frame parameters can be changed using the '*WinLength*' option (in second) and the '*Hop*' option (a value between 0 and 1).

mirsegment accepts uniquely as main input a ***miraudio*** objects not frame-decomposed, not channel decomposed, and not already segmented. Alternatively, **file name** or the ***'Folder'*** keyword can be used as well.

mirsegment(..., 'Novelty') can return several outputs:

1. the segmented audio waveform itself,
2. the novelty curve (***mirnovelty***) after peak picking (***mirpeaks***),
3. the similarity matrix (***mirsimatrix***), and
4. the features entered into the ***mirsimatrix*** operator.

mirmean

Description

mirmean(f) returns the mean along frames of the feature *f*.

If *f* is decomposed into segments, the results is the series of means in successive segments.

f can be a structure array composed of features. In this case, the output will be structured the same way.

mirstd

Description

mirmean(f) returns the standard deviation along frames of the feature *f*.

f can be a structure array composed of features. In this case, the output will be structured the same way.

mirstat

Description

mirstat can be applied to any object and will return its statistics in a structure array.

- If the object is frame-decomposed, the fields of the output are:
  - *Mean*: the average along frames;
  - *Std*: the standard deviation along frames;
  - *Slope*: the linear slope of the trend along frames, i.e. the derivative of the line that would best fit the curve. The slope *S* is computed in a normalised representation of the curve *C* – i.e., centered, with unit variance, and with a temporal scale reduced to a series *T* of values between 0 and 1 – as a solution in a least-square sense of the equation *S***T* = *C*;
  - *PeriodFreq*: the frequency (in Hz.) of the maximal periodicity detected in the frame-by-frame evolution of the values, estimated through the computation of the autocorrelation sequence. The first descending slope starting from zero lag is removed from this analysis, as it is not related to the periodicity of the curve. If no periodicity is detected – i.e., if there is absolutely no peak in the autocorrelation sequence –, *NaN* is returned;
  - *PeriodAmp*: the normalized amplitude of that main periodicity, i.e., such that the autocorrelation at zero lag is identically 1.0.
  - *PeriodEntropy*: the Shannon entropy of the autocorrelation function (cf. *mirentropy*).
- If the object is not frame-decomposed, the data itself is returned directly in the single field *Mean*.

An additional field *FileNames* indicates the file name following the same order used when displaying each numerical result.

NaN-filter

mirstat automatically discard any *NaN* value contained in the input data.

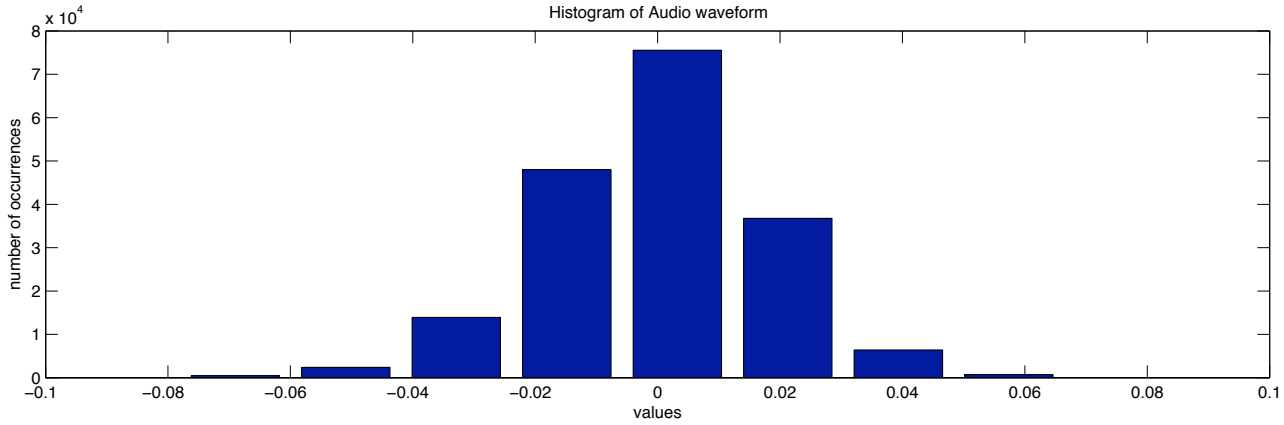
Management of structure arrays

If the input is already a structure array, the output will follows the same field decomposition (and all subfields as well) of the structure array, and will replace each final node with its corresponding *mirstat* structure array result.

mirhisto

Description

mirhisto can be applied to any object and will return its corresponding histogram. The data is binned into equally spaced containers.



Options

- *mirhisto(..., '**Number**', n)* specifies the number of containers. Default value : n = 10.
- *mirhisto(..., '**Ampli**')* adds the amplitude of the elements, instead of simply counting then.

Accessible Output

cf. §5.2 for an explanation of the use of the *get* method. Specific fields:

- '***Weight***': the number of elements associated to each container (same as '*Data*'),
- '***Bins***': the range of value associated to each bin (same as '*Pos*'). A first layer of values (along the third matrix dimension) indicates the minimal value for each bin, a second layer indicates the maximal values.

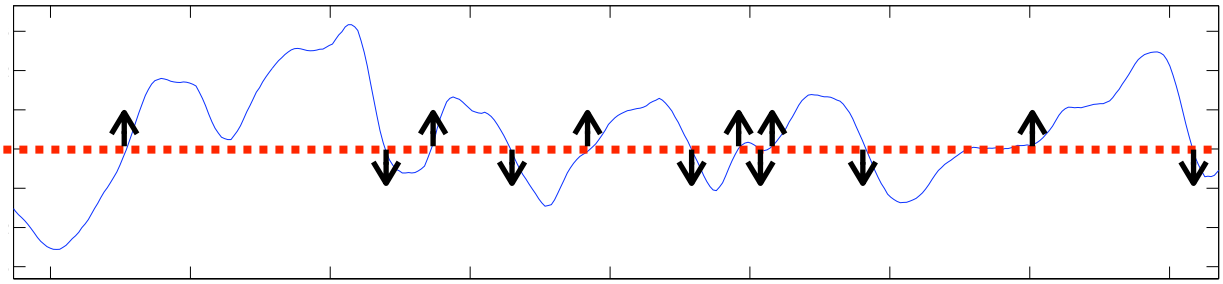
mirzerocross

DESCRIPTION

mirzerocross counts the number of times the signal crosses the X-axis (or, in other words, changes sign).

This function has already defined in as : applied directly to audio waveform, *mirzerocross* is an indicator of noisiness.

But actually *mirzerocross* accepts any input data type.



OPTIONS

- *mirzerocross*(..., '**Per**', *p*) precises the temporal reference for the rate computation. Possible values:
  - *p* = *'Second'*: number of sign-changes per second (Default).
  - *p* = *'Sample'*: number of sign-changes divided by the total number of samples. The '*Second*' option returns a result equal to the one returned by the '*Sample*' option multiplied by the sampling rate.
- *mirzerocross*(..., '**Dir**', *d*) precises the definition of sign change. Possible values:
  - *d* = *'One'*: number of sign-changes from negative to positive only (or, equivalently, from positive to negative only). (Default)
  - *d* = *'Both'*: number of sign-changes in both ways. The '*Both*' option returns a result equal to twice the one returned by the '*One*' option.

mircentroid

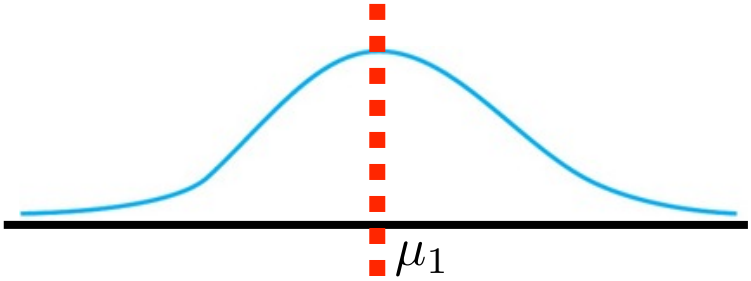
Description

mircentroid returns the *centroid* of the data.

Explanation

An important and useful description of the shape of a distribution can be obtained through the use of its moments. The *first moment*, called the *mean*, is the geometric center (*centroid*) of the distribution and is a measure of central tendency for the random variable.

$$\mu_1 = \int x f(x) dx$$



Inputs

Any data can be used as input.

If the input is an audio waveform, a file name, or the '*Folder*' keyword, the centroid is computed on the spectrum (spectral centroid).

If the input is a series of peak lobes produced by *mirpeaks(..., 'Extract')*, the centroid will be computed for each of these lobes separately.

Option

- When the input contains peaks (using ***mirpeaks***), *mircentroid(..., '**Peaks**', i)* will compute the centroids of the distribution of peaks. The argument *i* accepts two arguments:
  - *i* = *'NoInterpol'*: the centroid is computed on the non-interpolated peaks (default choice),
  - *i* = *'Interpol'*: the centroid is computed on the interpolated peaks (cf. '*Interpol*' option in *mirpeaks*).

mirspread

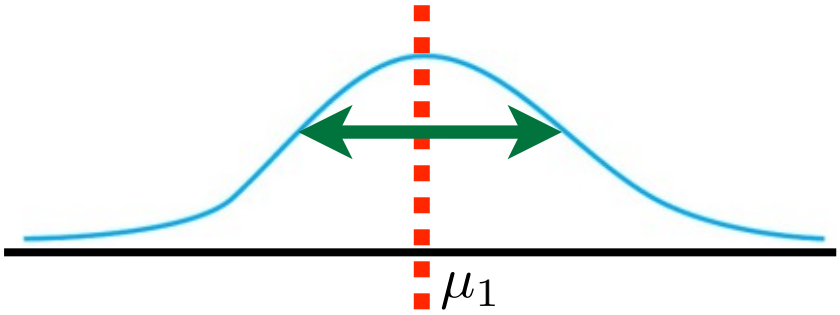
DESCRIPTION

mirspread returns the *standard deviation* of the data.

EXPLANATION

The *second central moment*, called the *variance*, is usually given the symbol sigma squared and is defined as:

$$\sigma^2 = \mu_2 = \int (x - \mu_1)^2 f(x) dx$$



Being the squared deviation of the random variable from its mean value, the variance is always positive and is a measure of the dispersion or spread of the distribution. The square root of the variance is called the *standard deviation*, and is more useful in describing the nature of the distribution since it has the same units as the random variable. (Koch)

INPUTS

Any data can be used as input.

If the input is an audio waveform, a file name, or the '*Folder*' keyword, the spread is computed on the spectrum (spectral spread).

If the input is a series of peak lobes produced by *mirpeaks(..., 'Extract')*, the spread will be computed for each of these lobes separately.

mirskewness

DESCRIPTION

mirskewness returns the *coefficient of skewness* of the data.

EXPLANATION

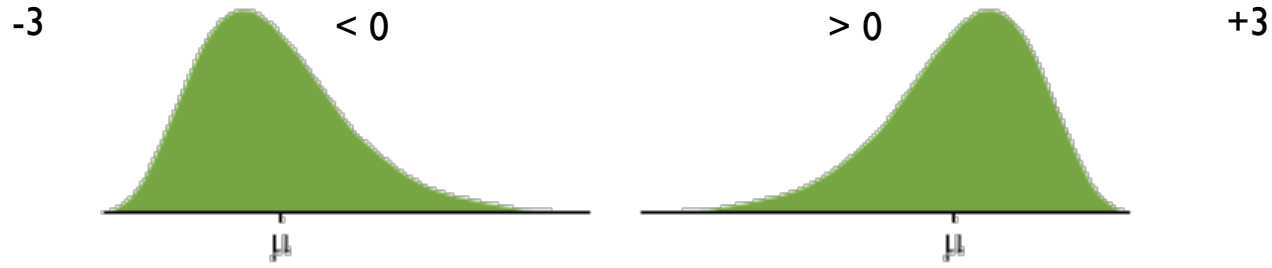
The *third central moment* is called the *skewness* and is a measure of the symmetry of the distribution. The skewness can have a positive value in which case the distribution is said to be positively skewed with a few values much larger than the mean and therefore a long tail to the right. A negatively skewed distribution has a longer tail to the left. A symmetrical distribution has a skewness of zero. (Koch)

$$\mu_3 = \int (x - \mu_1)^3 f(x) dx$$

The *coefficient of skewness* is the ratio of the skewness to the standard deviation raised to the third power.

$$\frac{\mu_3}{\sigma^3}$$

The coefficient of skewness has more convenient units than does the skewness and often ranges from -3.0 to 3.0 for data from natural systems. Again, a symmetrical distribution has a coefficient of skewness of zero. A positive coefficient of skewness often indicates that the distribution exhibits a concentration of mass toward the left and a long tail to the right whereas a negative value generally indicates the opposite. (Koch)



INPUTS

Any data can be used as input.

If the input is an audio waveform, a file name, or the '*Folder*' keyword, the skewness is computed on the spectrum (spectral skewness).

If the input is a series of peak lobes produced by *mirpeaks(..., 'Extract')*, the skewness will be computed for each of these lobes separately.

mirkurtosis

DESCRIPTION

mirkurtosis returns the (*excess*) *kurtosis*, of the data.

EXPLANATION

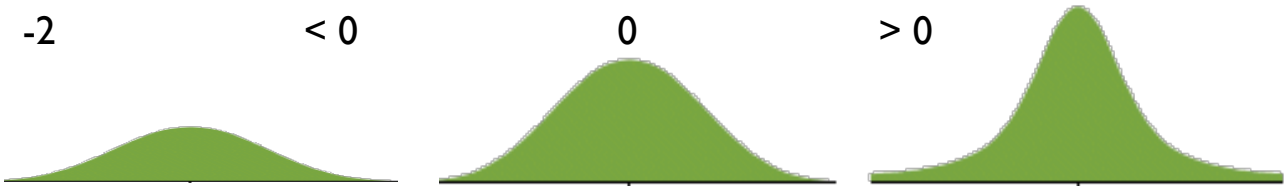
The *fourth standardized moment* is defined as,

$$\frac{\mu_4}{\sigma^4}$$

Kurtosis is more commonly defined as the fourth cumulant divided by the square of the variance of the probability distribution, equivalent to:

$$\frac{\mu_4}{\sigma^4} - 3$$

which is known as *excess kurtosis*. The "minus 3" at the end of this formula is often explained as a correction to make the kurtosis of the normal distribution equal to zero. Another reason can be seen by looking at the formula for the kurtosis of the sum of random variables. Because of the use of the cumulant, if Y is the sum of n independent random variables, all with the same distribution as X, then $Kurt[Y] = Kurt[X]/n$, while the formula would be more complicated if kurtosis were simply defined as *fourth standardized moment.* (Wikipedia)



INPUTS

Any data can be used as input.

If the input is an audio waveform, a file name, or the '*Folder*' keyword, the kurtosis is computed on the spectrum (spectral kurtosis).

If the input is a series of peak lobes produced by *mirpeaks(..., 'Extract')*, the kurtosis will be computed for each of these lobes separately.

mirflatness

Description

mirflatness returns the *flatness* of the data.

Explanation

The *flatness* indicates whether the distribution is smooth or spiky, and results from the simple ratio between the geometric mean and the arithmetic mean:

$$\frac{\sqrt[N]{\prod_{n=0}^{N-1} x(n)}}{\left(\frac{\sum_{n=0}^{N-1} x(n)}{N}\right)}$$

Inputs

Any data can be used as input.

If the input is an audio waveform, a file name, or the '*Folder*' keyword, the flatness is computed on the spectrum (spectral flatness).

mirentropy

Description

mirentropy returns the relative Shannon (1948) entropy of the input. The Shannon entropy, used in information theory, is based on the following equation:

$$H(X) := -\sum_{i=1}^{n} p(x_i) \log_b p(x_i)$$

where *b* is the base of the logarithm.

In order to obtain a measure of entropy that is independent on the sequence length, *mirentropy* actually returns the relative entropy, computed as follows:

*H(p) = -sum(p.*log(p)) /log(length(p));*

Shannon entropy offers a general description of the input curve *p*, and indicates in particular whether it contains predominant peaks or not. Indeed, if the curve is extremely flat, corresponding to a situation of maximum uncertainty concerning the output of the random variable *X* of probability mass function $p(x_i)$, then the entropy is maximal. Reversely, if the curve displays only one very sharp peak, above a flat and low background, then the entropy is minimal, indicating a situation of minimum uncertainty as the output will be entirely governed by that peak.

The equation of Shannon entropy can only be applied to functions $p(x_i)$ that follow the characteristics of a probability mass function: all the values must be non-negative and sum up to 1. Inputs of *mirentropy* are transformed in order to respect these constraints:

- The non-negative values are replaced by zeros (i.e., half-wave rectification).
- The remaining data is scaled such that it sums up to 1.

Inputs

Any data can be used as input.

If the input is an audio waveform, a file name, or the '*Folder*' keyword, the entropy is computed on the spectrum (spectral entropy).

Option

- *mirentropy(..., **'Center'**)* centers the input data before half-wave rectification.

mirfeatures

Description

mirfeatures computes a large set of features, and returns them in a structure array. If the result is stored in a variable *f*, for instance, then the features are organized as follows:

- in a *dynamics* field,
  - *f.dynamics.rms{1}*: the frame-based RMS (*mirrms*), that can be accessed using the command:
- in a *rhythm* field,
  - a *fluctuation* field, containing:
    - *f.rhythm.fluctuation.peak{1}*: a fluctuation summary (*mirfluctuation*) with its highest peak (*mirpeak*),
    - *f.rhythm.fluctuation.centroid{1}*: the centroid (*mircentroid*) of the fluctuation summary;
  - a *tempo* field, containing:
    - *f.rhythm.tempo{1}*: a frame-based tempo estimation (*mirtempo*),
    - *f.rhythm.tempo{2}*: the autocorrelation function used for the tempo estimation;
  - an *attack* field, containing:
    - a *time* field, with:
      - *f.rhythm.attack.time{1}*: the attack times (*mirattacktime*) of the onsets,
      - *f.rhythm.attack.time{2}*: the envelope curve used for the onset detection (*mironsets*);
    - a *slope* field, with:
      - *f.rhythm.attack.slope{1}*: the attack slopes (*mirattackslope*) of the onsets;
- in a *timbre* field,
  - *f.timbre.zerocross{1}*: the frame-decomposed zero-crossing rate (*mirzerocross*),
  - *f.timbre.centroid{1}*: the frame-decomposed spectral centroid (*mircentroid*),
  - *f.timbre.brightness{1}*: the frame-decomposed brightness (*mirbrightness*),
  - *f.timbre.spread{1}*: the frame-decomposed spectral spread (*mirspread*),
  - *f.timbre.skewness{1}*: the frame-decomposed spectral skewness (*mirskewness*),

- *f.timbre.kurtosis{1}*: the frame-decomposed spectral kurtosis (*mirkurtosis*),
- *f.timbre.rolloff95{1}*: the frame-decomposed roll-off (*mirrolloff*), using a 95 % threshold,
- *f.timbre.rolloff85{1}*: the frame-decomposed roll-off, using a 85 % threshold,
- *f.timbre.spectentropy{1}*: the frame-decomposed spectral entropy (*mirentropy*),
- *f.timbre.flatness{1}*: the frame-decomposed spectral flatness (*mirflatness*),
- a *roughness* field, containing:
  - *f.timbre.roughness{1}*: the frame-decomposed roughness (*mirroughness*),
  - *f.timbre.roughness{2}*: the spectrogram, containing the peaks used for the roughness estimation;
- an *irregularity* field, containing:
  - *f.timbre.irregularity{1}*: the frame-decomposed irregularity (*mirregularity*),
  - *f.timbre.irregularity{2}*: the spectrogram, containing the peaks used for the irregularity estimation;
- an *inharmonicity* field, containing:
  - *f.timbre.inharmonicity{1}*: the frame-decomposed inharmonicity (*mirinharmonicity*),
  - *f.timbre.inharmonicity{2}*: the spectrogram used for the inharmonicity estimation;
- *f.timbre.mfcc{1}*: the frame-decomposed MFCCs (*mirmfcc*),
- *f.timbre.dmfcc{1}*: the frame-decomposed delta-MFCCs,
- *f.timbre.ddmfcc{1}*: the frame-decomposed delta-delta-MFCCs,
- a *lowenergy* field, containing:
  - *f.timbre.lowenergy{1}*: the low energy rate (*mirlowenergy*),
  - *f.timbre.lowenergy{2}*: the RMS energy curve used for the low energy rate estimation;
- *f.spectralflux{1}*: the frame-decomposed spectral flux (*mirflux*);

- in a *pitch* field,
  - *f.pitch.salient{1}*: the frame-decomposed pitches (*mirpitch*),
  - a *chromagram_* field, containing:

    - *f.pitch.chromagram.peak{1}*: an unwrapped chromagram (*mirchromagram*) and its highest peak,
    - *f.pitch.chromagram.centroid{1}* field: the centroid of the chromagram;
- in a *tonal* field,
    - *f.tonal.keyclarity{1}*: the frame-decomposed key clarity (second output of *mirkey*),
    - *f.tonal.mode{1}*: the frame-decomposed mode (*mirmode*),
    - *f.tonal.hcdf{1}*: the frame-decomposed HCDF (*mirhcdf*).

Option

- *mirfeatures(..., '**Stat**')* returns the statistics of the features instead of the complete features themselves. In this way, the complete results are not accumulated in the RAM, preventing memory shortage problems.
- *mirfeatures(..., '**Segment**', t)* segments the audio sequence at the temporal positions indicated in the array *t* (in s.), and analyzes each segment separately.

mirmap

When *mirmap* is used for academic research, please cite the following publications:

Olivier Lartillot, Tuomas Eerola, Petri Toiviainen, Jose Fornari, "Multi-feature modeling of pulse clarity: Design, validation, and optimization", *International Conference on Music Information Retrieval*, Philadelphia, 2008.

DESCRIPTION

mirmap(predictors_file, ratings_file) performs a statistical mapping between ratings associated to a set of audio recordings, and a set of variables computed for the same set of audio recordings. It might be useful in particular when the set of predictors is large and diverse.

- The predictors were saved in one or several text file(s) whose name *predictors_file* (or a cell array of file name) is given as first argument of the *mirmap* function. This text file is the result of the exportation (using *mirexport*) of a chosen analysis performed using *MIRtoolbox*, on the folder of audio files.
- The ratings are saved in a one or several text file(s) whose name *ratings_file* (or a cell array of file name) is given as second argument of the *mirmap* function. This text file contains a simple column of values, with one value for each file analyzed in the folder of audio files under study, using the same ordering of files as for the predictors, i.e., a lexicographic order of file name.

The routine detects all features not sufficiently normal, based on a Lilliefors test (using Statistics Toolbox's *lillietest*) at the 5% significance level. Those non-normal features are then normalized using an optimization algorithm that automatically finds optimal Box-Cox transformations () of the data, ensuring that their distributions become sufficiently Gaussian, which is a prerequisite for correlation estimation. Features hence normalized that are still not considered as normal (this times using a more tolerant significance level of 1%) are then excluded from further analyses. They are listed in the Command Window under the heading "*Excluded:*".

From the selected features, only those whose correlation with the ratings is sufficiently statistically significant (with a p-value lower than .05) are selected.

The selected features are ordered from the most correlated to the least correlated ones. Features that are not sufficiently independent with respect to the better scoring ones (with a normalized crosscorrelation exceeding .6) are deleted as well.

EXAMPLE

f = mirfeatures('Folder')

mirexport('mypredictors.txt', f)

mirmap('mypredictors.txt', 'myratings.txt')

Output

mirmap returns a structure array, with the following fields:

- *normal*: the correlation between the normalized features and the ratings:
  - *cor*: the correlation value,
  - *pval*: the related p value;
- *significant*: the correlation between the statistically significant features and the ratings:
  - *cor*: the correlation value,
  - *pval*: the related p value,
  - *inter*: the cross-correlations with the other features,
  - *fields*: the feature name,
  - the resulting *alpha* and *lambda* values of the Box-Cox optimization;
- *best*: the correlation between the independent best features and the ratings:
  - *index*: the indices of chosen features among the significant ones,
  - *fields*: the feature name,
  - the related Box-Cox *alpha* and *lambda* values.

Text Display

Various information are output in the Command Window:

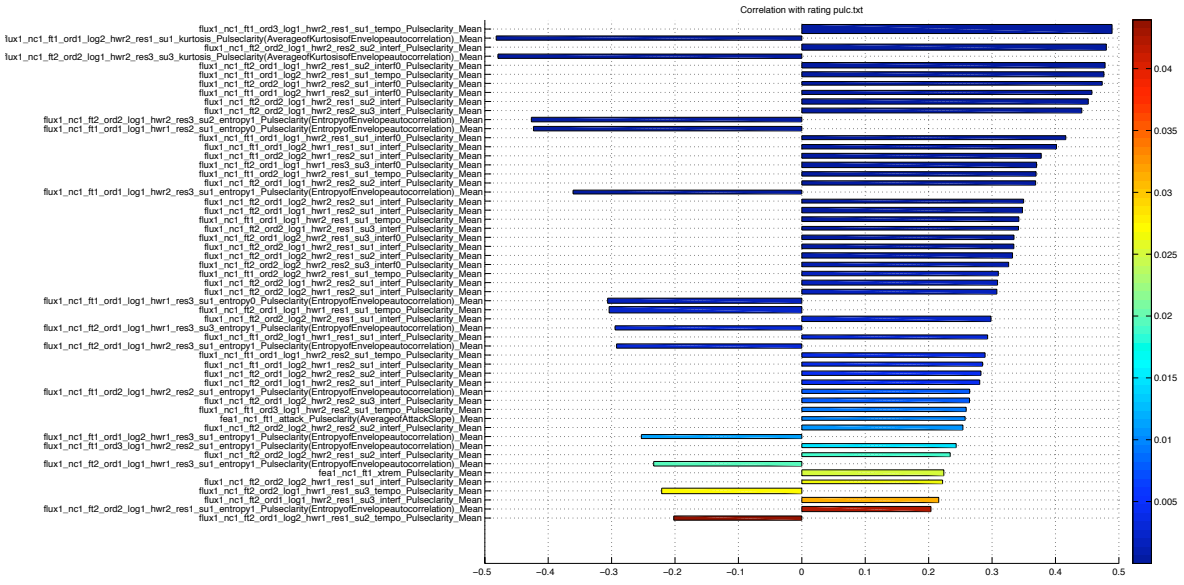
- the excluded features (cf. below)
- the finally selected features are then displayed, including:
  - the index among the significant features (*best.index*),
  - the feature name,
  - the correlation value with respect to the rating,
  - the worse cross-correlation with respect to better rating features.

- the result of a step-wise regression (using Statistics Toolbox *stepwisefit*), showing the successive steps, and the final results.
- the result of a normalized step-wise regression, this time centering and scaling each feature (using *stepwisefit* option '*scale*').
- The list of features included in the final regression.

Graphical Output

The correlation obtained by each of the best independent features are represented by a series of horizontal bars with best results at the top and worse results at the bottom. The abscissa axis represents the correlation values, that can be either positive or negative. The width of each bar indicates the independence of each feature with respect to the higher ones, and is computed as one minus the worse normalized cross-correlation. Finally, the color indicates the p-value, following the color scale given by the colorbar on the right of the figure: cold colors for statistically significant correlations, and warm colors for correlations of limited significance.

The figure below show a real-life example of useful application of the *mirmap* script. A large set of predictors were computed and stored in a highly structured array, composed of a hierarchical set of cells and structures corresponding to various methods for the predictor estimation.



miremotion

The current version of *miremotion* is calibrated with version 1.3. Its use with more recent versions 1.3.1, 1.3.2 and 1.3.3 gives distorted results.

When *miremotion* is used for academic research, please cite the following publication:

Tuomas Eerola, Olivier Lartillot, Petri Toiviainen, "Prediction of Multidimensional Emotional Ratings in Music From Audio Using Multivariate Regression Models", *International Conference on Music Information Retrieval*, Kobe, 2009.

Description

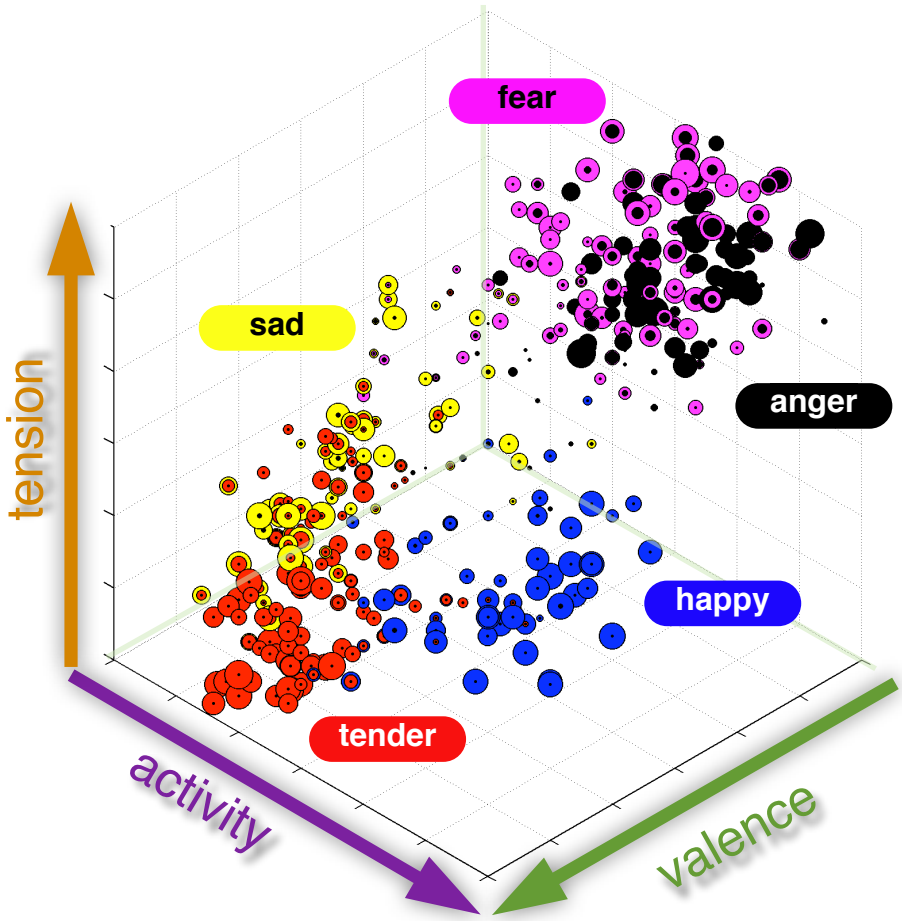
Emotions evoked in music is usually described along two opposite paradigms (cf. Eerola, Lartillot, Toiviainen, 2009 for a literature review and a study of their interdependencies):

- Either as a decomposition into a few basic emotions. In *miremotion*, the 5 classes are happy, sad, tender, anger, fear.
- Either as a multi-dimensional space where the three dimensions, used in *miremotion*, are activity (or energetic arousal), valence (a pleasure-displeasure continuum) and tension (or tense arousal).

miremotion attempts to predict such description of emotion based on the analysis of the audio and musical contents of the recordings. Hence the output of *miremotion* corresponds to this underlying localization of emotional content within the 5 basic classes and within the 3 dimensions.

Each class or dimension is supposed to have values spanned in the interval [1,7]. Value 1 would correspond to very low value, and value 7 to very high value. This convention is used because the models are based on listeners' emotional ratings that were collected exactly the same way, as it corresponds to a classical Libert scale used in experimental psychology.

But even if the model was constructed using observation spanning in the interval [1,7], particular audio examples, not considered in the training, can extend beyond that range. So the interval [0,8] looks like a more probable range of value for these dimensions and concepts.



Comparison between basic concepts of emotions (happy, sad, tender, anger, fear) and emotion dimensions (activity, valence, tension) (Eerola, Lartillot and Toiviainen, 2009).

Predictive Models

Following a systematic statistical methodology, a mapping has been found between each emotion concept or dimension and various sets of audio and musical features. The details of the procedure is given in (Eerola, Lartillot and Toiviainen, 2009).

The implemented models are based on multiple linear regression with 5 best predictors (MLR option in the paper). In this first version of *miremotion,* the Box-Cox transformations have been removed until the normalization values have been established with a large sample of music.

- The five factors contributing to the **activity** score are:
  - RMS averaged along frames (β = 0.6664)
  - Maximum value of summarized fluctuation (β = 0.6099)
  - Spectral centroid averaged along frames (β = 0.4486)

    - Spectral spread averaged along frames (β = -0.4639)
    - Entropy of the smoothed and collapsed spectrogram, averaged along frames (β = 0.7056)
- The five factors contributing to the **valence** score are:
    - Standard deviation of RMS along frames (β = -0.3161)
    - Maximum value of summarized fluctuation (β = 0.6099)
    - Key clarity averaged along frames (β = 0.8802)
    - Mode averaged along frames (β = 0.4565)
    - Averaged spectral novelty (β = 0.4015)
- The five factors contributing to the **tension** score are:
    - Standard deviation of RMS along frames (β = 0.5382)
    - Maximum value of summarized fluctuation (β = -0.5406)
    - Key clarity averaged along frames (β = -0.6808)
    - Averaged HCDF (β = 0.8629)
    - Averaged novelty from unwrapped chromagram (β = -0.5958)
- The five factors contributing to the **happy** score are:
    - Maximum value of summarized fluctuation (β = 0.7438)
    - Spectral spread averaged along frames (β = -0.3965)
    - Standard deviation of the position of the maximum of the unwrapped chromagram (β = 0.4047)
    - Key clarity averaged along frames (β = 0.7780)
    - Mode averaged along frames (β = 0.6620)
- The five factors contributing to the **sad** score are:
    - Spectral spread averaged along frames (β = 0.4324)
    - Standard deviation of the position of the maximum of the unwrapped chromagram (β = -0.3137)
    - Mode averaged along frames (β = -0.5201)

- Averaged HCDF (β = -0.6017)
- Averaged novelty from wrapped chromagram (β = 0.4493)

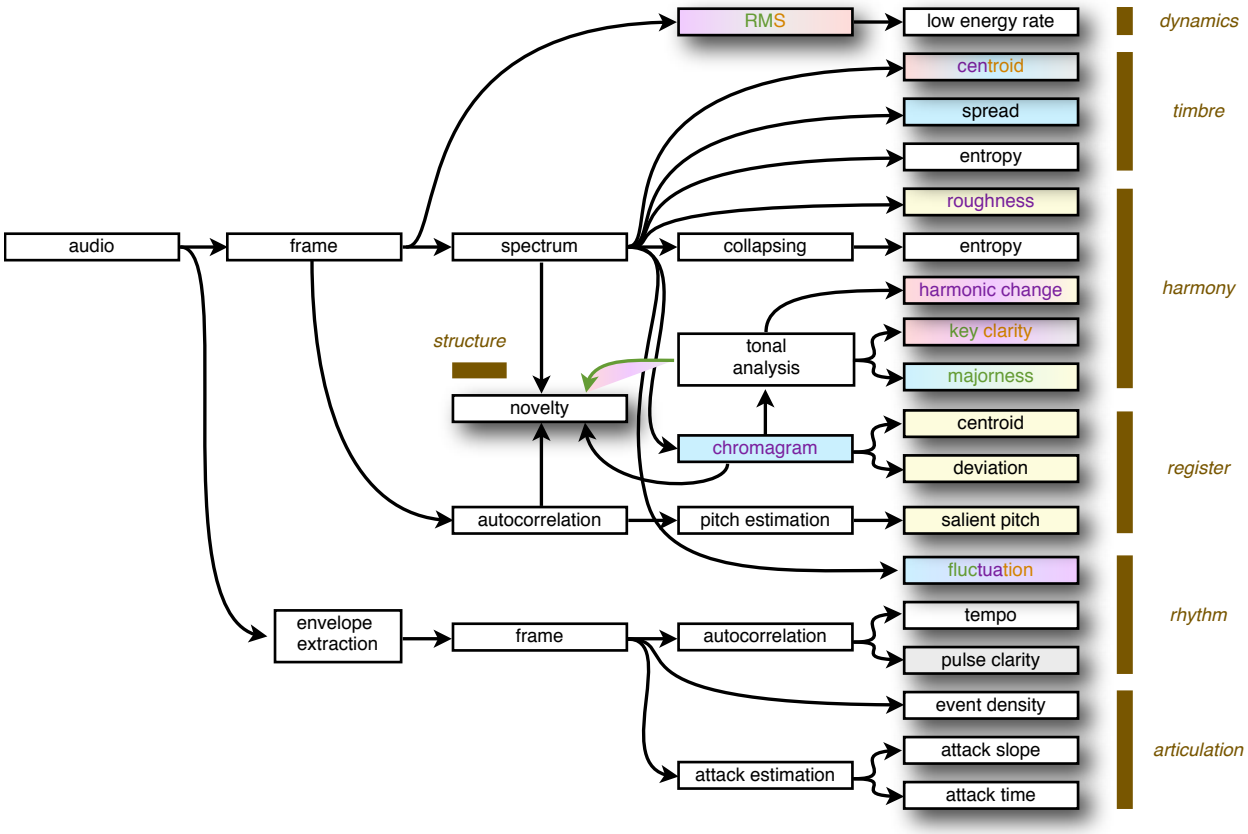
- The five factors contributing to the **tender** score are:
  - Spectral centroid averaged along frames (β = -0.2709)
  - Standard deviation of roughness (β = -0.4904)
  - Key clarity averaged along frames (β = 0.5192)
  - Averaged HCDF (β = -0.3995)
  - Averaged spectral novelty (β = 0.3391)
- The five factors contributing to the **anger** score are:
  - Roughness averaged along frames (β = 0.5517)
  - Key clarity averaged along frames (β = -0.5802)
  - Entropy of the smoothed and collapsed spectrogram, averaged along frames (β = 0.2821)
  - Averaged novelty from unwrapped chromagram (β = -0.2971)
- The five factors contributing to the **fear** score are:
  - Standard deviation of RMS along frames (β = 0.4069)
  - Averaged attack time (β = -0.6388)
  - Maximum value of summarized fluctuation (β = -0.2538)
  - Key clarity averaged along frames (β = -0.9860)
  - Mode averaged along frames (β = -0.3144)

For later version of *MIRtoolbox,* we plan to revise the coefficients in order to

(a) force the output range between 0 - 1 and

(b) ground them on alternative models and materials (training sets).

A classifier of emotion concepts based also on audio features is also under development.



Audio and musical features founding the prediction of emotion. The colors associated to each feature corresponds to the correlating emotions, following the same color code that in the previous figure (Eerola, Lartillot and Toiviainen, 2009).

Optional arguments

- *miremotion(..., 'Frame', l, h)* predicts the emotional content for each successive frame of length *l*, and with a hop factor of *h*. By default, *l* = 1 second and no overlapping.
- *miremotion(..., 'Dimensions', n)* indicates the number of emotion dimensions to output (0, 2 or 3(. *By default, n* = 3. Alternatively, these dimensions can be listed:
  - *miremotion(..., 'Activity')*
  - *miremotion(..., 'Valence')*
  - *miremotion(..., 'Tension')*

A value *n* = 2 will output Activity and Valence only.

- *miremotion(..., 'Arousal')* corresponds to *miremotion(..., 'Activity', 'Tension')*.

- *miremotion(..., 'Concepts')* outputs all the emotion concepts (chosen by default). Alternatively, these concepts can be listed:
  - *miremotion(..., 'Happy')*
  - *miremotion(..., 'Sad')*
  - *miremotion(..., 'Tender')*
  - *miremotion(..., 'Anger')*
  - *miremotion(..., 'Fear')*
- *miremotion(..., 'Concepts', 0)* excludes all emotion concepts in the results.

Accessible Output

cf. §5.2 for an explanation of the use of the *get* method. Specific fields:

- '***Dim***': the list of emotion dimensions taken into consideration,
- '***DimData***': the score associated with each of these emotion dimensions,
- '***Class***': the list of emotion classes taken into consideration,
- '***ClassData***': the score associated with each of these emotion classes,
- '***ActivityFactors***': the value associated with each of the factors contributing to the activity score (in the same order as presented above),
- '***ValenceFactors***': the value associated with each of the factors contributing to the valence score (in the same order as presented above),
- '***TensionFactors***': the value associated with each of the factors contributing to the tension score (in the same order as presented above),
- '***HappyFactors***': the value associated with each of the factors contributing to the happy score (in the same order as presented above),
- '***SadFactors***': the value associated with each of the factors contributing to the sad score (in the same order as presented above),
- '***TenderFactors***': the value associated with each of the factors contributing to the tender score (in the same order as presented above),
- '***AngerFactors***': the value associated with each of the factors contributing to the anger score (in the same order as presented above),

- '***FearFactors***': the value associated with each of the factors contributing to the fear score (in the same order as presented above).

mirclassify

DESCRIPTION

mirclassify(test, features_test, train, features_train) classifies the audio sequence(s) contained in the audio object *test*, along the analytic feature(s) *features_test*, following the supervised learning of a training set defined by the audio object *train* and the corresponding analytic feature(s) *features_train*. Multiple analytic features have to be grouped into one array of cells.

Train samples need to be labelled with their respective classes using the '*Label*' option in *miraudio*. If test samples are labelled as well, the correctness of the classification is evaluated.

Requires the *Netlab* toolbox.

You can also integrate your own arrays of numbers computed outside *MIRtoolbox* as part of the features. These arrays should be given as matrices where each successive column is the analysis of each successive file.

EXAMPLE

mirclassify(test, mfcc(test), train, mfcc(train))

mirclassify(test, {mfcc(test), centroid(test)}, train, {mfcc(train), centroid(train)})

OPTIONAL ARGUMENTS

- *mirclassify(..., '**Nearest**')* uses the minimum distance strategy. (by default)
- *mirclassify(..., '**Nearest**', k)* uses the *k*-nearest-neighbour strategy. Default value: *k* = 1, corresponding to the minimum distance strategy.
- *mirclassify(..., '**GMM**', ng)* uses a gaussian mixture model. Each class is modeled by at most *ng* gaussians. Default value: *ng* = 1.
  - Additionnally, the type of mixture model can be specified, using the set of value proposed in *Netlab*'s *gmm* function: i.e., '*spherical*', '*diag*', '*full*' (default value) and '*ppca*'. (cf. help *gmm*)

ACCESSIBLE OUTPUT

cf. §5.2 for an explanation of the use of the *get* method. Specific fields:

- '***Classes***': the class associated to each test sample (same as '*Data*'),
- '***Correct***': the correctness of the classification, from 0 to 1 (available only if test samples have been labelled too).

mircluster

DESCRIPTION

- *mircluster(a, f, ...)*, where *a* is one or several audio files *already segmented* (using *mirsegment*), and *f* is an analysis of the same data, clusters the segments in the audio sequence(s) contained in the audio object *a*, along the analytic feature(s) *f*, using the *k*-means strategy.
  - The analytic feature(s) *f* should *not* be frame decomposed. Frame-decomposed data should first be summarized, using for instance *mirmean* or *mirstd*.
  - Multiple analytic features have to be grouped into one array of cells.
- *mircluster(d, ...)*, where *d* is any *frame-decomposed* feature computed using *MIRtoolbox*, clusters the segments in the audio sequence(s) contained in the audio object *a*, along the analytic feature(s) *f*, using the *k*-means strategy. Multiple analytic features have to be grouped into one array of cells.

mircluster simply calls the *kmeans_clusters* function from the *SOM toolbox* (that is part of *MIRtoolbox* distribution) with the data contained in the *MIRtoolbox* objects provided as input. This is the only use of *SOM toolbox* in *MIRtoolbox*.

EXAMPLE

- Clustering of audio segments:

```
sg = mirsegment(a);

mircluster(sg, mirmfcc(sg))

mircluster(sg, {mirmfcc(sg), mircentroid(sg)})
```

- Clustering of frames:

```
cc = mirmfcc(a,'Frame');

mircluster(cc)
```

OPTIONAL ARGUMENT:

- *mircluster(..., n)* indicates the maximal number of clusters. Default value: *n* = 2.
- *mircluster(..., '**Runs**', r)* indicates the maximal number of runs. Default value: *r* = 5.

Accessible Output

cf. §5.2 for an explanation of the use of the *get.* method. Specific fields:

- '***Clusters***': a structured descriptions of the clustered reduction (centroids, reduction of the initial representation as a sequence of clusters, etc.),

4.4. Similarity and Retrieval

Operators have been added for the comparison between audio files and for query by examples.

mirdist

DESCRIPTION

mirdist(x,y) evaluates the distance between audio files along a particular representation specified by the user, corresponding to audio or musical features computed using *MIRtoolbox*. The input variables *x* and *y* are the obtained representations computed for the chosen audio files.

x should corresponding to the representation of one particular file, whereas *y* can correspond to the representation of either one particular file:

```
x = mirmfcc('file1.mp3');

y = mirmfcc('file2.mp3');

mirdist(x,y)
```

or a folder of files:

```
x = mirmfcc('file1.mp3');

y = mirmfcc('Folder');

mirdist(x,y)
```

- If *x* and *y* are not decomposed into frames,

  mirdist(..., metric) specifies the distance method *metric*, following the same list as used in Matlab *pdist* command (cf. *help pdist*). Default value: *metric* = *'Cosine'*.

- If *x* and *y* are composed of clustered frames (using *mircluster*), the cluster signatures are compared using Earth Mover Distance (Logan and Salomon, 2001).

- If x and y contains peaks, the vectors representing the peak distributions are compared using Euclidean distance (used with *mirnovelty* in Jacobson, 2006).

mirquery

DESCRIPTION

mirquery(q,b), where

- *q* is the analysis of one audio file and
- *b* is the analysis of a folder of audio files,

according to the same *MIRtoolbox* feature, returns the name of the audio files in the database *b* in an increasing distance to *q* with respect to the chosen feature, where the distance is computed using *mirdist*.

mirquery(d), where *d* is the distance between one audio file and a folder of audio file, according to a *MIRtoolbox* feature, returns the name of the audio files in an increasing distance *d*.

OPTIONAL ARGUMENTS

- *mirquery(..., **'Best'**, n)* returns the name of the *n* closest audio files.
- *mirquery(.., **'Distance'**, metric)* specifies the distance method *metric* to use (cf. *mirdist*). Default value: *metric* = *'Cosine'*.

The successive results can then be played using *mirplay*.

mirgetdata

Description

mirgetdata return the data contained in the input in a structure that can be used for further computation outside *MIRtoolbox*.

Output Format

- If the input is based on one non-segmented audio sequence, the result is returned as a matrix. The columns of the matrix usually correspond to the successive frames of the audio signal. The third dimension of the matrix corresponds to the different channels of a filterbank.
- If the input corresponds to a set of audio sequences, and if each sequence has same number of frames, the corresponding resulting matrices are concatenated columnwise one after the other. If the number of raws of the elementary matrices varies, the missing values are replaced by NaN in the final matrix. On the contrary, if the number of columns (i.e., frames) differs, then the result remains a cell array of matrices.
- Idem if the input corresponds to one or several segmented audio sequence(s).

Input and Output

- If the input is a key strength curve, the fourth dimension distinguishes between major and minor keys. i.e.:
  - *d(:,:,:,1)* is the keystrength for the major keys, and
  - *d(:,:,:,2)* is the keystrength for the minor keys.
- If the input is a key estimation, two output are returned:
  a. the keys (from 1 to 12) and
  b. the modes (1 for major, 2 for minor).
- If the input is the result of a peak detection, two output are returned:
  a. the position of the peaks and
  b. the value corresponding to these peaks, in the units predefined for this data.

What is returned by *mirgetdata* is no more a *MIRtoolbox* object, but just a *Matlab* array (or matrix). So of course, if you display such array using *Matlab plot* function, what you get is a curve without any proper X-axis annotation. (So by default the X-axis simply indicates the sample indices).

mirexport

mirexport(filename, ...) exports statistical information related to diverse data into a text file called filename.

mirexport('Workspace', ...) instead directly output the statistical information in a structure array saved in the Matlab workspace. This structure contains three fields:

- *filenames*: the name of the original audio files,
- *types*: the name of the features,
- *data*: the data.

The exported data should be related to the same initial audio file or the same ordered set of audio files.

The data listed after the first arguments can be:

- any feature computed in *MIRtoolbox*. What will be exported is the statistical description of the feature (using the *mirstat* function)
- any structure array of such features. Such as the ouput of the *mirstat* function.
- any cell array of such features.
- the name of a text file. The text file is imported with the Matlab *importdata* command. Each line of the file should contains a fixed number of data delimited by tabulations. The first line, or 'header', indicates the name of each of these columns.

The file format of the output can be either:

- a text file. It follows the same text file representation as for the input text files. The first column of the matrix indicates the name of the audio files. The text file can be opened in Matlab, or in a spreadsheet program, such as Microsoft Excel, where the data matrix can be automatically reconstructed.
- an attribute-relation file. It follows the ARFF standard, used in particular in the WEKA data mining environment.

If the audio files were initially labelled using the '*Label*' option in *miraudio*, then these labels are automatically indicated as classes in the ARFF output file.

5. ADVANCED USE OF *MIRTOOLBOX*

5.1. Interface preferences

MIRPARALLEL (BETA)

mirparallel(1) toggles on parallel processing: when '*Folder*' or '*Folders*' is used, several audio files can be analysed in parallel using several parallel *Matlab* sessions running on the different processors and/or processor cores of your computer. (Requires MathWorks' *Parallel Computing Toolbox*.)

mirparallel(0) toggles back off parallel processing.

MIRVERBOSE

mirverbose(0) toggles off the display by *MIRtoolbox* of minor informations in the Matlab Command Window (such as "*Computing mirfunction ...*").

mirverbose(1) toggles back on the display of such information.

MIRWAITBAR

mirwaitbar(0) toggles off the display by *MIRtoolbox* of waitbar windows.

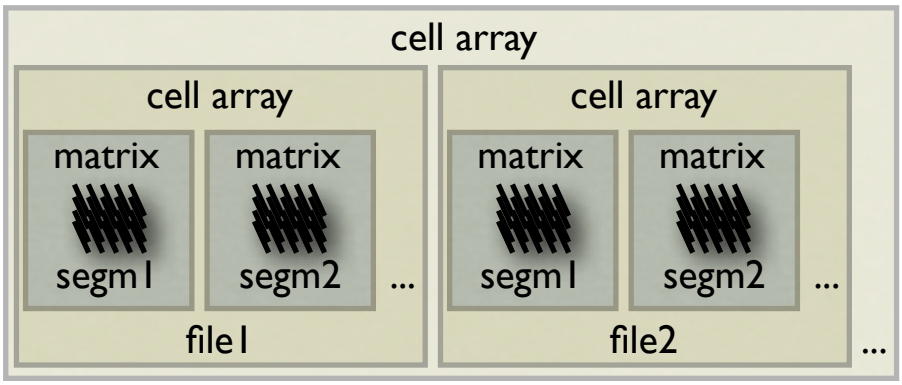
mirverbose(1) toggles back on the display of these waitbar windows.

5.2. get

get returns fields of *MIRtoolbox* objects.

General fields used by most objects are the following:

- '***Data***': the computed values, stored in a cell array, where each cell, corresponding to one audio file, is itself a cell array, where each cell, corresponding to one segment, is a matrix, where columns indicate successive frames, and where multiple channels are represented in the 3rd dimension.



- '***Pos***': the X-axis abscissae related to the Y-axis ordinates given by *'Data'*, and stored in the same way as for '*Data*'.
- '***Unit***': the name of the unit in which the values are represented,
- '***Sampling***': the sampling rate of the data, stored in a array of numbers, one number for each audio file.
- '***Length***': the duration of the input audio signal, in number of samples.
- '***NBits***': the resolutions in number of bits of the initial audio file, stored in the same ways as for '*Sampling*'.
- '***Name***': the name of the audio files, stored in a cell array, where each cell indicates the name of one file.
- '***Title***': the name of the data type,
- '***Channels***': the number and indexing of the channels decomposing the initial signal, stored in a cell array, where each cell contains the channel indexes for one audio file.

The list of more specific fields are indicated in the "*Accessible Output*" paragraph concluding the description of each feature.

5.3. Memory management

There are important things to know in order to take benefit of the memory management mechanism offered by *MIRtoolbox*.

AUTOMATED MEMORY OPTIMIZATION

When a *MIRtoolbox* operator is called with '*Folder*' (or '*Folders*') as main argument, the command is applied to each audio file in the current directory one file after the other. The previous audio file is cleared from the memory when the next audio file is considered. The results for all audio file are combined into the main memory and returned. For that reason, '*Folder*' (and '*Folders*') keywords should be applied to the feature of highest possible level. For instance, when evaluating the key of a batch of file,

```
o = mirkey('Folder')
```

each audio file is processed one after the other, and only the key result of each file is stored and grouped with those of the other files. This does not cause any memory problem.

On the contrary, the following set of commands should be avoided:

```
a = miraudio('Folder')

o = mirkey(a)
```

Why is this code problematic? Because after executing the first line, *all* the waveform of *all* the audio files are supposed to be stored together in the memory, which might sometimes leads to memory overflow problems.

Similarly, when a *MIRtoolbox* operator is called with used for a particularly long audio file, the audio file is automatically decomposed into little chunks that can fit into the memory. The command is applied to each successive chunk one after the other. The previous chunk is cleared from the memory when the next chunk is considered. The results for all chunks are combined into the main memory and returned. For that reason, audio files should be applied to the feature of highest possible level. For instance, when evaluating the key evolution of long file,

```
o = mirkey('myhugefile', 'Frame')
```

each successive chunk is processed one after the other, and only the key result of each chunk is stored and concatenated with those of the other chunks. This does not cause any memory problem.

On the contrary, the following set of commands should be avoided:

```
a = miraudio('myhugefile')

o = mirkey(a, 'Frame')
```

Why is this code problematic? Because after executing the first line, the *complete* waveform of the audio files is supposed to be stored entirely in the memory, which might sometimes leads to memory overflow problems.

For those reasons, as mentioned previously, when possible avoid writing a succession of operators if you can write in one line more efficiently.

Hopefully, there is a way to avoid such consideration by using the '*Design*' keyword, as will be explained below.

Flowchart design

If you write this command:

```
a = miraudio('myhugefile')

s = mirspectrum(a, 'Frame')

c = mircentroid(s)
```

there may be memory problem when computing the spectrogram. In order to benefit from *MIRtoolbox* memory management mechanisms, you should write instead:

```
a = miraudio('Design')

s = mirspectrum(a, 'Frame')

c = mircentroid(s)

mireval(c, 'myhugefile')
```

Complex flowchart design

But if you want to get several output in your flowchart, for instance:

s = mirspectrum('mysong');

cent = mircentroid(s);

ceps = mircepstrum(s);

You should use a *mirstruct* object, which store all the outputs of your flowchart into fields and write as follows:

myflow = ***mirstruct***

All the temporary data used in your flowchart should be stored into a *tmp* field. Hence when a spectrogram is used for several final features, it should be a temporary variable:

myflow = ***mirstruct***

*myflow****.tmp****.s = mirspectrum('****Design****', 'Frame');*

*myflow.cent = mircentroid(myflow****.tmp****.s);*

*myflow.ceps = mircepstrum(myflow****.tmp****.s);*

output = ***mireval****(myflow, 'myhugefile');*

Similarly, when an onset detection curve is used for several final features, it should be a temporary variable:

myflow = ***mirstruct****;*

*myflow****.tmp****.o = mironsets('****Design****');*

*myflow.at = mirattacktime(myflow****.tmp****.o);*

*myflow.as = mirattackslope(myflow****.tmp****.o);*

myflow = mirstat(myflow);

*output = **mireval**(myflow, 'myhugefile');*

Please note also that in the current version of *MIRtoolbox*, it is not possible to identify directly an output variable with a temporary variable, such as:

*r.rms = r.dynamics**.tmp.rms**; % Does not work yet.*

As a temporary solution, you can call the *MIRtoolbox* operator once again, such as:

*r.rms = mirrms(r.dynamics**.tmp.rms**); % This is OK.*

Note also that, as can be seen in the previous example, you can keep only the statistics of the results instead of the complete results by adding the line:

myflow = mirstat(myflow);

STRUCTURED COMPLEX FLOWCHART DESIGN

You can decompose your flowchart into several fields, using the standard structure array used in *Matlab*, for instance associating each field with one particular musical dimension (such as '*dynamics*', etc.). You can assign a temporary variable within one particular field, by declaring that field as a *mirstruct* object:

*r.dynamics = **mirstruct**;*

*r.dynamics**.tmp.rms** = mirrms(a, 'Frame');*

*r.dynamics.mean = mirmean(r.dynamics**.tmp.rms**);*

*r.dynamics.lowenergy = mirlowenergy(r.dynamics**.tmp.rms**, 'ASR');*

*r.timbral = **mirstruct**;*

...

mireval(r, 'Folders');

In this case, the temporary variable (here, *r.dynamics.tmp.rms*) should be used only inside that particular field in which it has been defined (here, *r.dynamics*).

REFERENCES

Alonso, David, Richard. A study of tempo tracking algorithms from polyphonic music signals, 4 COST 276 Workshop, 2003.

Juan P. Bello, Chris Duxbury, Mike Davies, and Mark Sandler, On the Use of Phase and Energy for Musical Onset Detection in the Complex Domain, *IEEE Signal Processing Letters*, 11-6, 2004.

Paul Boersma (1993). Accurate short-term analysis of the fundamental frequency and the harmonics-to-noise ratio of a sampled sound. IFA Proceedings 17: 97-110.

Paul Boersma & David Weenink (2005). *Praat: doing phonetics by computer* [Computer program] http://www.praat.org/

B. P. Bogert, M. J. R. Healy, and J. W. Tukey: The quefrency alanysis of time series for echoes: cepstrum, pseudo-autocovariance, cross-cepstrum, and saphe cracking. *Proceedings of the Symposium on Time Series Analysis* (M. Rosenblatt, Ed) Chapter 15, 209-243. New York: Wiley, 1963.

T. Eerola, O. Lartillot, P. Toiviainen, "Prediction of Multidimensional Emotional Ratings in Music From Audio Using Multivariate Regression Models", *International Conference on Music Information Retrieval*, Kobe, 2009.

H. Fastl. Fluctuation strength and temporal masking patterns of amplitude-modulated broad-band noise. *Hearing Research*, 8:59–69, 1982.

Y. Feng, Y. Zhuang, Y. Pan. (2003) Popular music retrieval by detecting mood, *ACM SIGIR Conference on Research and Development in Information Retrieval*.

J. Foote, M. Cooper, U. Nam. (2002) "Audio Retrieval by Rhythmic Similarity", *ISMIR 2002*.

Foote, Cooper. (2003). Media Segmentation using Self-Similarity Decomposition. *SPIE Storage and Retrieval for Multimedia Databases*, 5021, 167-75.

Gómez, E. (2006). *Tonal description of music audio signal*. Phd thesis, Universitat Pompeu Fabra, Barcelona .

Harte, C. A. and M. B. Sandler, Detecting harmonic change in musical audio, in *Proceedings of Audio and Music Computing for Multimedia Workshop*, Santa Barbara, CA, 2006.

A. Klapuri, "Sound onset detection by applying psychoacoustic knowledge", *IEEE International Conference on Acoustics, Speech, and Signal Processing*, 1999.

Klapuri, A., A. Eronen and J. Astola. (2006). "Analysis of the meter of acoustic musical signals", IEEE Transactions on Audio, Speech and Langage Processing, 14-1, 342– 355.

R. Koch, A Tutorial in Probability and Statistics, http://web.cecs.pdx.edu/-roy/tutorial/Stat_int.htm

Krimphoff, J., McAdams, S. & Winsberg, S. (1994), Caractérisation du timbre des sons complexes. II : Analyses acoustiques et quantification psychophysique. *Journal de Physique*, 4(C5), 625-628.

Krumhansl, *Cognitive foundations of musical pitch*. Oxford UP, 1990.

K. Jacobson, *A multifaceted approach to music similarity*, ISMIR 2006.

Jensen, *Timbre Models of Musical Sounds*, Rapport 99/7, University of Copenhagen, 1999.

Juslin, P. N. (2000). Cue utilization in communication of emotion in music performance: relating performance to perception. *Journal of Experimental Psychology: Human Perception and Performance*, 26(6), 1797-813.

Olivier Lartillot, Tuomas Eerola, Petri Toiviainen, Jose Fornari, "Multi-feature modeling of pulse clarity: Design, validation, and optimization", International Conference on Music Information Retrieval, Philadelphia, 2008.

Laukka, P., Juslin, P. N., and Bresin, R. (2005). A dimensional approach to vocal expression of emotion. *Cognition and Emotion*, 19, 633-653.

Logan, B., A. Salomon. (2001). A content-based music similarity function. Cambridge Research Laboratory, Technical Report Series.

McAulay, R.; Quatieri, T. (1996). “Speech analysis/Synthesis based on a sinusoidal representation”, Acoustics, Speech and Signal Processing, IEEE Transactions on, Volume 34, Issue 4. Page(s): 744 - 754

Nabney, I. ”NETLAB: Algorithms for pattern recognition”, *Springer Advances In Pattern Recognition Series*, 2002.

Pampalk, E. “A Matlab Toolbox to Compute Similarity from Audio”, *International Conference on Music Information Retrieval*, 2004

E. Pampalk, A. Rauber, D. Merkl, "Content-based Organization and Visualization of Music Archives", *ACM Multimedia* 2002, pp. 570-579.

R. D. Patterson et al. “Complex sounds and auditory images,” in *Auditory Physiology and Perception*, Y. Cazals et al, Oxford, 1992, pp. 429-446.

Peeters. G. (2004). *A large set of audio features for sound description (similarity and classification) in the CUIDADO project*. version 1.0

Peeters. Music pitch representation by periodicity measures based on combined temporal and spectral representations. *ICASSP* 2006.

Plomp & Levelt "Tonal Consonance and Critical Bandwidth" *Journal of the Acoustical Society of America*, 1965.

Pohle, T., E. Pampalk and G. Widmer (2005). Evaluation of Frequently Used Audio Features for Classification of Music Into Perceptual Categories. *Proceedings of the Fourth International Workshop on Content-Based Multimedia Indexing (CBMI'05)*, Riga, Latvia, June 21-23.

Scheirer, E. D. (1998). “Tempo and beat analysis of acoustic musical signals”, *Journal of the Acoustical Society of America*, 103-1, 588–601.

Sethares, W. A. (1998). *Tuning, Timbre, Spectrum, Scale*, Springer-Verlag.

Shannon, C. E. 1948. “A Mathematical Theory of Communication.” *Bell Systems Technical Journal* 27:379–423.

Slaney, M. *Auditory Toolbox Version 2*, Technical Report. Interval Research Corporation, 1998-010, 1998.

S.S. Stevens and J. Volkman (1940) "The relation of pitch to frequency: A revised scale" *Am. J. Psychol.* 53: 329-353.

Terhardt, E. Calculating virtual pitch. *Hearing Research*, 1:155–182, 1979.

Toiviainen & Krumhansl, “Measuring and modeling real-time responses to music: The dynamics of tonality induction”, *Perception* 32-6, pp. 741–766, 2003.

Toiviainen, Snyder. “Tapping to Bach: Resonance-based modeling of pulse”, *Music Perception*, 21-1, 43–80, 2003.

Tolonen, Karjalainen. A Computationally Efficient Multipitch Analysis Model, *IEEE Transactions on Speech and Audio Processing*, 8(6), 2000.

Tzanetakis, Cook. Musical genre classification of audio signals. *IEEE Tr. Speech and Audio Processing*, 10(5),293-302, 2002.

Van Noorden, L., & Moelants, D. (1999). Resonance in the Perception of Musical Pulse. *Journal of New Music Research*, 28(1), 43-66.

Vassilakis, P. N. (2001). Perceptual and Physical Properties of Amplitude Fluctuation and their Musical Significance. Doctoral Dissertation. Los Angeles: University of California, Los Angeles; Systematic Musicology.

Vesanto, J. ”Self-Organizing Map in Matlab: the SOM Toolbox”, *Matlab DSP Conference*, 35–40, 1999.

E. Zwicker, G. Flottorp and S.S. Stevens (1957) "Critical bandwidth in loudness summation" *J. Acoust. Soc. Am.* 29: 548-557.